修士学位論文

リアルタイムアプリケーションのための階層的空間分割手法に関する研究

平成 21 年度
先端表現情報学 コース
086302
今村 紀之
指導教員　河口洋一郎 教授

概要

コンピュータ グラフィクスをはじめとする情報科学分野においては，空間を扱うシステムが数多く存在する．レイ トレーシング法や物理シミュレーション，パターン認識はその一例である．これらの応用に際しては，適切な空間探索手法の開発がアプリケーション全体の効率性を左右する．

一方，ハードウェアにおける進歩とともにこれらのアプリケーションのリアル タイム化が進んでいる．近年のハードウェアの進化の傾向として，演算装置単体の処理能力の向上から複数のコアやプロセッサを用いた並列実装による性能の向上を目指す方向へシフトしてきており，リアルタイム化に際しては並列処理に適したアルゴリズムを用いた手法の開発が不可欠である．

これらの背景を受け，本研究では並列環境に適した，リアルタイム アプリケーションのための空間分割手法の提案を行う．

効率的な階層的空間分割手法の一つとして Bounding Volume Hierarchy (BVH) があげられる．これは，primitive を Axis-Aligned Bounding Box (AABB) など単純な形状に置き換え，木構造を形成したものである．BVH の高速な並列構築手法として空間充填曲線の一種である Morton 曲線を利用した手法 (Linear BVH; LBVH) が存在する．しかし，Morton 曲線を特徴づける Morton コードは一様格子を利用して付与されるため，空間を均一に離散化しており空間的局在性を十分に考慮にいれることができない．そのため，primitive の大きさの分布にばらつきがある場合など，シーンによっては BVH の品質が極端に低下してしまうことがある．

そこで，本手法では空間に存在する primitive の分布を，ヒストグラムを用いて離散化し，Morton コードを生成するための格子をゆがめることにより，局在性を反映させる．ヒストグラムの対象として，各 primitive における AABB の中心点や表面積，体積などを用いることができる．

本研究では，提案手法をトップ ダウンにより構築した BVH (Space Median BVH; MBVH) および LBVH と比較した．構築速度は LBVH にくらべ 10 % 以下のオーバーヘッドに抑えら

れた一方，生成された木構造においては，そのリーフの深さ分布が改善され，シーンによっては MBVH に匹敵する品質の BVH を生成することができた．

これらの手法を実際にレイ トレーシング法へ組み込み，その性能の検証を行った．その性能はシーンによって大きく異なるものの，最適なヒストグラム生成法を選ぶことで，5〜40 % のレンダリング速度向上を達成することができるという結果になった．シーンにおける最適なヒストグラム生成法をどのようにして選択するかの指標を提示することが今後の課題としてあげられる．

目次

第 1 章　序論 1

1.1　はじめに 1

1.1.1　情報科学における空間の扱い 1

1.1.2　情報科学におけるリアルタイム性の意義 2

1.1.3　本論文の構成 2

1.2　背景 3

1.2.1　計算幾何学における諸問題 3

1.2.2　コンピュータ グラフィクスと計算幾何アルゴリズム 7

1.2.3　リアルタイムアプリケーションに向けて 17

1.3　目的 19

第 2 章　関連研究 21

2.1　空間分割データ構造に関する研究 21

2.1.1　uniform grid 21

2.1.2　octree 23

2.1.3　BSP 木と kd-tree 24

2.1.4　BV と BVH 28

2.2　本研究の位置づけ 32

第 3 章　DBVH の提案 33

3.1　LBVH 33

3.2　DBVH の概要 35

3.3　DBVH のアルゴリズム 36

第 4 章　DBVH の実装 38

4.1 CPU における DBVH の実装 38
4.2 ヒストグラム歪曲化の派生的手法の実装 42

第 5 章 DBVH のリアルタイムアプリケーションに向けた応用 45
5.1 レイ トレーシング法への応用 45
5.1.1 光線の表現と ray-scene intersection test 45
5.1.2 BVH の traversal 46
5.1.3 ray-AABB テスト 48

第 6 章 結果と考察 51
6.1 空間分割手法の比較実験 51
6.1.1 空間分割手法の実験設定 51
6.1.2 実験モデル データ 52
6.1.3 実験環境 52
6.2 構築速度による比較と考察 52
6.3 品質による比較と考察 56
6.4 レイ トレーシング法への応用による比較 60

第 7 章 結論 76
7.1 結論 76
7.2 今後の展望 76
7.3 あとがき 77

謝辞 78

参考文献 81

付録 A 略語一覧 92

図目次

1.1 交差判定の例 4
1.2 2 次元と 3 次元での点群に対する凸包 4
1.3 Delaunay 三角形分割と Voronoi 図 7
1.4 フラクタルを用いて生成した hight field による地形 8
1.5 CSG による表現 9
1.6 物理ベース モデリングを用いた CG 表現 10
1.7 Delaunay 三角形分割および四面体分割を利用して生成した FEM のための非構造メッシュの例 11
1.8 レイ トレーシング法の模式図 12
1.9 代表的なレンダリング手法とその流れ 14
1.10 MCPT, BDPT, MLT の模式図 15
1.11 フォトン マッピング法の 1pass 目の概略図 16
1.12 レイ トレーシング法とフォトン マッピング法のレンダリング画像の比較 . . . 18

2.1 uniform grid の概略図 22
2.2 DDA 法の 2 次元での例 23
2.3 quadtree の概略図 24
2.4 5D octree の模式図 25
2.5 BSP の概略図と BPS 木 26
2.6 object median 戦略による kd-tree の構築 27
2.7 改変 sliding midpoint split による kd-tree の構築 27
2.8 BV の例 (2 次元) 29
2.9 BV の例 (3 次元) 30
2.10 2 次元 BVH のトップダウン構築 31

3.1 2 次元の Morton コード 34
3.2 LBVH の構築手法 35
3.3 DBVH の概念図 36
3.4 prefix scan の例 37

4.1 DBVH 構築のフローチャート 39

5.1 BVH traversal のフロー チャート 47
5.2 AABB 48
5.3 2 次元での slab test の例 49

6.1 実験に用いたモデル データ 53
6.2 chess モデルにおける空間分割構造の構築時間の比較 54
6.3 knot モデルにおける空間分割構造の構築時間の比較 54
6.4 buddha モデルにおける空間分割構造の構築時間の比較 55
6.5 uzu モデルにおける空間分割構造の構築時間の比較 55
6.6 chess モデルに対する MBVH の構築結果 57
6.7 chess モデルに対する LBVH の構築結果 58
6.8 chess モデルに対する DBVH の構築結果 59
6.9 chess モデルにおけるリーフ ノードの深度分布の比較 61
6.10 knot モデルにおけるリーフ ノードの深度分布の比較 61
6.11 buddha モデルにおけるリーフ ノードの深度分布の比較 62
6.12 uzu モデルにおけるリーフ ノードの深度分布の比較 62
6.13 chess モデルに対する MBVH，LBVH，DBVH のリーフノードの深さ分布 . . 63
6.14 knot モデルに対する MBVH，LBVH，DBVH のリーフノードの深さ分布 . . 64
6.15 レンダリングの結果画像 65
6.16 chess シーン (1 次光線のみ) の traversal での時間比較 66
6.17 chess シーン (1 次光線 + 影光線) の traversal での時間比較 66
6.18 MBVH での走査したノード数の表示 (chess シーン) 67
6.19 LBVH での走査したノード数の表示 (chess シーン) 67
6.20 DBVH での走査したノード数の表示 67
6.21 DBVH-AD での走査したノード数の表示 (chess シーン) 67
6.22 DBVH-OA での走査したノード数の表示 (chess シーン) 68

6.23 DBVH-SA での走査したノード数の表示 (chess シーン) 68
6.24 DBVH-VL での走査したノード数の表示 (chess シーン) 68
6.25 カラー バー 68
6.26 maxplanck シーン (1 次光線のみ) の traversal での時間比較 70
6.27 maxplanck シーン (1 次光線 + 影光線) の traversal での時間比較 70
6.28 MBVH での走査したノード数の表示 (maxplanck シーン) 71
6.29 LBVH での走査したノード数の表示 (maxplanck シーン) 71
6.30 DBVH での走査したノード数の表示 71
6.31 DBVH-AD での走査したノード数の表示 (maxplanck シーン) 71
6.32 DBVH-OA での走査したノード数の表示 (maxplanck シーン) 72
6.33 DBVH-SA での走査したノード数の表示 (maxplanck シーン) 72
6.34 DBVH-VL での走査したノード数の表示 (maxplanck シーン) 72
6.35 カラー バー 72
6.36 spontza シーン (1 次光線のみ) の traversal での時間比較 73
6.37 spontza シーン (1 次光線 + 影光線) の traversal での時間比較 73
6.38 MBVH での走査したノード数の表示 (spontza シーン) 74
6.39 LBVH での走査したノード数の表示 (spontza シーン) 74
6.40 DBVH での走査したノード数の表示 74
6.41 DBVH-AD での走査したノード数の表示 (spontza シーン) 74
6.42 DBVH-OA での走査したノード数の表示 (spontza シーン) 75
6.43 DBVH-SA での走査したノード数の表示 (spontza シーン) 75
6.44 DBVH-VL での走査したノード数の表示 (spontza シーン) 75
6.45 カラー バー 75

表目次

1.1 凸包アルゴリズムとその計算量 6

4.1 DBVH のノードのデータ構造の例 42

6.1 実験に用いたモデル データ 52

6.2 1 リーフに分けきれなかった primitive 数の比較 56

第 1 章

序論

1.1 はじめに

1.1.1 情報科学における空間の扱い

情報科学分野において,「空間」(space) を扱う機会は非常に多い[*1]．ヒルベルト空間 (Hilbert space)，アドレス空間 (address space)，名前空間 (namespace)，状態空間 (state space) など空間を冠する専門用語や概念が数多く存在する．

例えば，自然情報処理のような分野では，画像や音声といった情報を空間上に配置し，識別器によって空間上の情報をクラスタリング (clustering) し，パターン認識を行うことが常套的に行われている．この際，識別器として一般に用いられる，ニューラル ネットワーク (neural network) や SVM (Support Vector Machine; サポート ベクター マシン)，機械学習などの各種手法において，空間をどう効率的に扱うかという問題が深く関わってくる．また，数値科学計算の分野では，その目的に合わせて，空間上に存在する物理量の表現方法が多種多様に提案されており，その如何が精度および計算速度，スケーラビリティ (scalability) に密接に関わってくる．FEM (Finite Element Method; 有限要素法) では，空間上に要素[*2]を分布させ，その各頂点が物理量を保持するデータ構造になっている一方，粒子法などの point-based な手法では空

[*1] もちろん，情報科学以外の分野においても，空間に関する議論は多く行われる．例えば，天体物理や天文工学は宇宙「空間」(outer space) を扱う分野であり，相対性理論などに代表されるように古くから，空間と時間の関係性に関する研究が盛んに行われている．また，社会学や経済学においては，物理空間における経済・社会活動の差異に関して議論を行う際に，空間的不均等性や空間統合 (spatial integration) という概念がしばしば用いられ，経済地理学のような新たな学問分野の興りにつながるなど学問一般に「空間」に対する興味は非常に深いといえる．

[*2] 主に四面体や六面体が用いられる．

間上に分布する点そのものが物理量を保持しており，その用途や目的により手法の使い分けがなされている．中でも，CG (Computer Graphics; コンピュータ グラフィクス) 分野のような，計算幾何学と密接に関わる分野では，特に空間表現に関する研究が古くから盛んに行われている．

1.1.2　情報科学におけるリアルタイム性の意義

一方で，情報科学分野においては，CPU (Central Processing Unit; 中央処理装置) や GPU (Graphics Processing Unit; グラフィクス プロセシング ユニット) などにおけるプロセッサのマルチコア/メニーコア (multi-core/ many core) 化やコンピュータ クラスター (computer cluster) を利用した HPC (High-Performance Computing) 並列化などのハードウェアの急速な発展を受け，従来まででは考えられないような速度での演算処理が可能となってきている．これらを利用し，そのパフォーマンスを最大限に発揮することで，近年，前述の各種手法の多くが，リアルタイム (real-time; 実時間) に実現できるようになってきている．

計算時間の短縮による，アプリケーション (application) のリアルタイム化がもたらす恩恵は測り知れない．そもそも実世界からの物理的入力を受けて実世界へ出力を返すシステムの多くは，クリティカルなリアルタイム性が要求されることが非常に多く，リアルタイムと非リアルタイムはその用途の上で明確に分類されている．例えば，ロボット工学においては，刻々と変わる実世界をリアルタイムにセンシングし，その系を制御することで，システムは目的を達成する．自動車や工業機械の制御システムにおいては，待ち時間が許容されないことは明らかである．また，CAD (Computer Aided Design; コンピュータ支援設計)/ CAM (Computer Aided Manufacturing; コンピュータ支援製造) や AR (Augumented Reality; 拡張現実)，ゲームなどユーザとのインタラクションを必要とする情報システムにおいてもごくわずかな遅延がユーザのフラストレーションを誘発することになる．従来までであれば非リアルタイム用途にしか用いることができなかったアルゴリズムがこのようなリアルタイムアプリケーションに応用可能となることで，前述のようなタイム クリティカルなシステムにおいて実現できることの幅が格段に広がるのである．

1.1.3　本論文の構成

以上を踏まえ，本論文ではリアルタイム アプリケーションへの応用を意識し，効率的な空間表現に関する議論を行い，新たな空間分割手法の提案と実際の応用に際する考察を行う．

まず，本章では 1.2 節にて，空間分割データ構造に関する背景およびその応用例を紹介し，

本研究の目的を 1.3 節にて述べる．

第 2 章では，空間分割データ構造に関する先行研究について，主に CG 分野に主眼を当てて紹介する．2.2 節で本研究の位置づけを行い，以後の章でその具体的な方法論を提案する．

第 3 章では，本研究で提案する手法 DBVH (Distorted Bounding Volume Hierarchy) の概要を述べ，アルゴリズムに関する議論を行う．

さらに第 4 章でその実装方法を述べ，それを用いて第 5 章で，実際のアプリケーションへの応用を行う．今回は，アプリケーションとしてレイ トレーシング法を選択した．

これらの実装による結果は第 6 章にまとめたうえで，考察についても併せて行う．

本論文の結論は第 7 章にて述べる．

なお，本論文ではアルファベットによる略語を多用する．一部を除いて，初回の使用時にその正式名称及び日本語訳を併記する．これらの一覧は付録 A にまとめて掲載するので，必要に応じて参考にされたい．

1.2 背景

1.2.1 計算幾何学における諸問題

計算幾何学 (computational geometry) は 1971 年に Forrest[26] が提唱した概念で，計算機にて幾何学的問題を扱う学問である．Forrest はコンピュータを利用し，幾何学的に製品デザインを行うことを想定した数学表現を行っており，現在ではその概念は CAD/CAM という用語とともに，広く一般に応用されている．計算幾何学は，幾何学的な概念を扱う以上，空間に関する問題とは不可分であり，その応用分野としても，CG や地理情報処理，ロボット工学，VLSI (Very Large Scale Integration) 設計，CAE (Computer Aided Engineering; コンピュータ支援工学) など多岐に渡る．以来，幾何学的アルゴリズムは数多く提案され，今なお研究が行われている分野である．

本研究においては，これらの幾何学的アルゴリズムが主に対象とする，n 次元[*3]のユークリッド距離空間をその例として採用し，空間情報の表現に関して議論する．

以下に幾何学的アルゴリズムの問題例を記す．なお，計算量の尺度としてはランダウの O 記法 (Landau symbol) を用いた．

[*3] 基本的には次元は 3 次元として扱うが，その他の次元に対しても，基本的に同様に扱うことができる．ただし，ほとんどの問題において次元に対して指数関数的に複雑度が大きくなり，高次での問題を 3 次元同様の手法で解くことは非効率的であることが多い．この現象は一般に「次元の呪い」とよばれている．

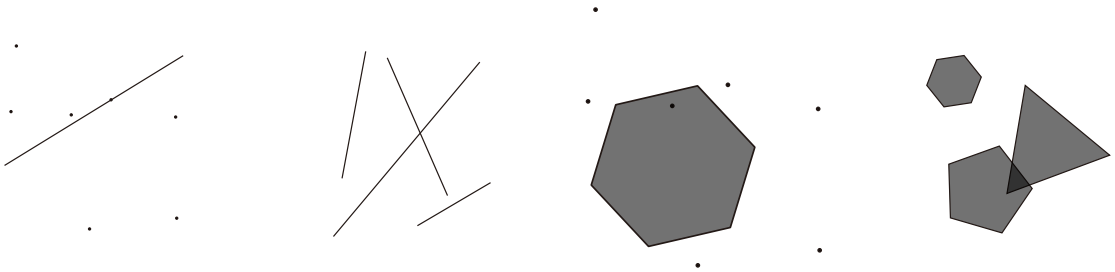


Fig. 1.1　交差判定の例．左から，a) 点と線分の交差判定，b) 線分と線分の交差判定，c) 点と多角形の交差判定，d) 多角形と多角形の交差判定．いづれも 2 次元での例．

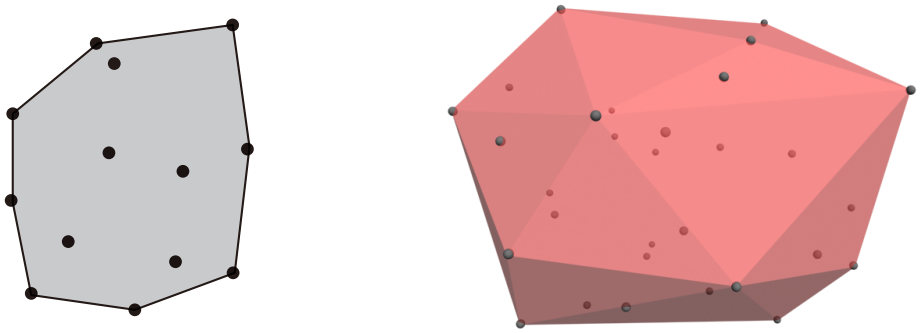


Fig. 1.2　2 次元 (左) と 3 次元 (右) での点群に対する凸包．n 次元空間の点群 $S \in \mathbb{R}^n$ に対して一意に定まる．

交差判定　空間上のある 2 つの集合が互いに交わるかどうかを判定する問題，すなわち，ある空間 U における集合 $A \subset U, B \subset U$ に対して，$A \cap B$ を求める問題である．A, B がどのような集合であるかによって，分類することができる．例えば，A，B ともに単一の線分である場合，A が線分，B が平面である場合，A が線分，B が多面体である場合，A, B ともに多面体である場合などその問題ごとに解法が異なる．

また，その応用によっては，$A \cap B$ そのものを求める必要がなく，その体積が必要な場合，重心位置さえわかればよい場合，空集合かどうかのみがわかればよい場合などもあり，その用途や精度などもどの解法を用いるかを左右する要因となりうる．(Fig.1.1) に交差判定の問題を例示する．

凸包　ある与えられた点の集合に対して，それらすべてを包含する最小の凸多角形および凸多

面体[*4]を求める問題である．実際に点群に対して凸包を求めた例を (Fig.1.2) に掲載する．凸包を求めるアルゴリズムとして，Quickhull[23, 7] や包装 (gift wrapping) 法 [13], 分割統治 (divide and conquer) 法 [88]，Andrew's monotone chain [1, 21] などが知られている．これらのアルゴリズムの計算量について (Table1.1) にまとめる．

Delaunay 三角形分割　空間 U 上の母点の集合 $S \subset U$ における，2 点 $p_i, p_j \in S$ は，p_i と p_j を通り，その内部や境界に S の他の母点を含まない円 C が存在するとき，かつその場に限り Delaunay 辺を形成する．すべての点についての Delaunay 辺によって構成されるグラフ DT(S) を Delaunay 三角形分割 (Delaunay 図) とよぶ．Delaunay 辺は Delaunay 三角形を構成しており，S の凸包は，すべて三角形で分割されることになる．このとき，各三角形の外接円は内部や境界に頂点 p_i，p_j，p_k 以外の点を含まないという特徴がある．3 次元の場合には Delaunay 四面体分割とよばれ，最適メッシュ生成アルゴリズムとして用いられることも多い．また，その双対グラフは Voronoi 図とよばれ，1644 年に René Descartes によってその概念がもたらされて以来，データ圧縮やワイヤレスネットワークの構築などに用いられている[*5]．実際の点群に対する Delaunay 三角形分割及び Voronoi 図の適用例を (Fig.1.3) に示す．Delaunay 三角形/四面体分割のアルゴリズムとしては，Bowyer-Watson のアルゴリズム [12, 111]，Fortune のアルゴリズム [27], k-means 法 [71] など，$O(n \log n)$ の計算量で求める手法が存在する．近年では，疑似的な Voronoi 図およびその応用手法が提案されている [86, 49].

幾何学的最適化問題　幾何学的最適化に関する問題にはいくつかあるが，ここでは例として，線形計画問題について述べる．

いくつかの 1 次不等式及び 1 次等式[*6]を満たす点の中で，ある 1 次関数[*7]を最小化もしくは最大化する点を求める問題を線形計画問題とよぶ．線形 (不) 等制約は実行可能領域とよばれる凸多面体を定義する．目的関数が線形であるため，最適解はこの凸多面体の境界に現れる．この性質を利用し，最適解に達するまで多面体の辺をたどってより高い目的関数の点を次々にたどることで線形計画問題を解く手法を単体法 (simplex method) とよぶ[*8]．その後，カーマーカー法や主双対内点法などの解法が提案されて

[*4] 空間が 2 次元の場合には凸多角形，3 次元の場合には凸多面体となる．これを凸包 (convex hull) とよぶ.

[*5] 気象学の分野では Thiessen 多角形として降雨量測定などで用いられているほか，物性物理や量子力学分野では結晶の結合エネルギーを求める際に Wigner-Seitz 胞という概念として用いられており，情報科学分野以外でも非常に重要な概念となっている.

[*6] これを線形 (不) 等制約とよぶ.

[*7] これを目的関数とよぶ.

[*8] 単体は幾何学の用語であり，0 次元では点，1 次元では線分，2 次元では三角形，3 次元では四面体,4 次元では

Table 1.1 凸包アルゴリズムとその計算量．n は入力となる点の数．h は出力となる凸包の頂点の数．$h \leq n$ となることが保証されている．

手法	計算量
総当たりによる強引な手法	$O(n^4)$
包装法 [13]	$O(nh)$
Quickhull [23, 7]	$O(nh)$
Graham Scan [38]	$O(n \log n)$
分割統治法 [88]	$O(n \log n)$
Andrew’s Monotone Chain [1, 21]	$O(n \log n)$
Marriage before Conquest [57]	$O(n \log h)$

いる．

地理的最適化問題 グラフ理論に密接に関連する問題で，巡回セールスマン問題 (traveling salesman problem, TNP) や最小木問題などがある．前者の巡回セールスマン問題は，点の集合と辺の移動コストが与えられたときにすべての点をめぐり，元の点に戻ってくる閉回路の最小コストを求める問題である[*9]．後者の最小木問題は，無向グラフ $G = (V, E)$ の各枝 $e \in E$ に重み $w(e)$ が与えられているときに，グラフ G 上において全点 V を点集合とし，木になっている部分グラフの重みの和が最小となる木[*10]を求める問題である．

幾何学的探索問題 前述の交差判定も幾何学的探索問題の一つであるが，そのほかにも最近傍探索 (nearest neighbor search, NNS) などが例としてあげられる．これは，距離空間における最も近い点を探す問題であり，k 番目に近い点までを求める問題は k 近傍探索 (k-nearest neighbor search, KNNS) とよばれる．

計算幾何学分野での発展とともに，上記のような問題に関して，1970 年代より多くの解法が提案されてきた．しかし，その多くが NP 困難な問題であるため，リアルタイム アプリケーションで許容しうる実行速度では，一定以上大きな問題を扱うことができないという問題がある．そこで，一定の誤差や局所最適解への収束を許容した，近似的解法が提案されている．焼きなまし法や遺伝的アルゴリズム (genetic algorithm, GA) などがその例である．

五胞体を表す．

[*9] この問題は NP 困難な問題として有名である．

[*10] これを最小木とよぶ．

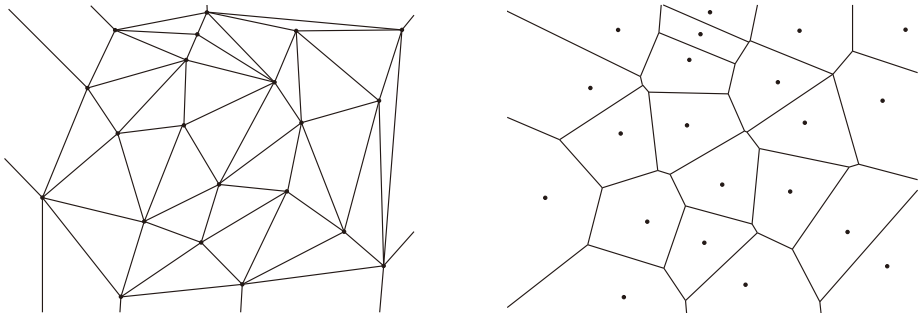


Fig. 1.3 Delaunay 三角形分割 (左) と Voronoi 図 (右). 同一の点群に対するグラフであるため，右のグラフと左のグラフは双対グラフの関係にあたる.

しかし，誤差が許容されない場合も多く，厳密解法に関しては，分枝限定法 (Branch and Bound, BB) とよばれる手法で，あらかじめ解の候補集合である解空間を階層的に分割し，効率よく探索する方法が多く提案されている.

1.2.2 コンピュータ グラフィクスと計算幾何アルゴリズム

CG は計算幾何と特に関わりの深い分野であり，前節で述べたような幾何アルゴリズムが多用される.

空間表現とモデリング

3DCG (3 Dimensional Computer Graphics; 3 次元コンピュータグラフィクス) 映像の制作は，一般に 3 次元空間上に物体をモデリング (modeling) する作業から始まる．モデリングにおいては，球や円柱といった単純形状 (primitive) や多数の多角形で構成されるポリゴン (polygon)，2 次曲線，NURBS (Non-Uniform Rational B-Spline; 非一様有理 B スプライン) 曲線等で構成される曲線・曲面，空間の濃度分布 (volume data; ボリューム データ) といった各種形状情報を空間上に定義し，シーンを形作っていく．1990 年頃から，計算機を用いたモデリングの特性を生かした方法として，シミュレーションを用いた[*11]モデリング手法が多く提案されている．その際に用いられる空間データ構造は，そのシミュレーション手法により多種多様な方法が存在する.

[*11] ここでは，いわゆるプロシージャル (procedual) とよばれる手法に関してもシミュレーションを用いたモデリング手法として紹介する．両者の区別に関してはっきりしたものはないが，CG ではシミュレーションという用語は「物理」を冠した物理シミュレーションに関して使われることが多く，それ以外をプロシージャルとよぶことが多い.



Fig. 1.4　フラクタルを用いて生成した hight field による地形．モデリング/レンダリング： Terragen.

例えば，地形を生成するためによく用いられる空間表現に height field がある (Fig.1.4)．これは，メッシュ (mesh) 上の計算点で高さを計算し，その高さの分だけメッシュを変位させる手法である．Musgrave ら [80] は，fBm (fractional Brownian motion; フラクタル ブラウン運動) とよばれる，フラクタルを用いた手法で，height field での隆起地形の自動生成手法を開発した．Lindstrom ら [69, 70] は TIN (Triangular Irregular Network) メッシュ[*12]とよばれる，四分木を元に階層的に細分化されたメッシュ上に height field を構築し，選択的に階層をレンダリングすることでリアルタイムに処理する手法を提案した．また，height field は簡易的に水の自由表面を表す手法としてもしばしば用いられる [83].

単純な図形を組み合わせて物体を作る CSG (Constructive Solid Geometry) もプロシージャル手法の一例である．プリミティブ同士の和や積といったブール演算を行い，ソリッド モデルを生成するモデリング手法である．CSG の実際の適用例を (Fig.1.5) に示す．この形状計算には，BSP (Binary Space Partitioning; 二分空間分割) 木を空間表現として用いることで効率化を図ることができる [81].

一方，1990 年後半ごろから物理シミュレーションを用いたモデリング手法が数多く提案されてきた．剛体，布，流体，弾性体，破壊，爆発など，対象となる現象は多岐にわたり，すでに映画やゲーム等の制作で実用されている．実際に物理シミュレーションを用いて破壊のモデリングを行った画像を (Fig.1.6) に示す.

[*12] そのほかにも 4-k メッシュや restricted triangle quadtree というよび方もある．

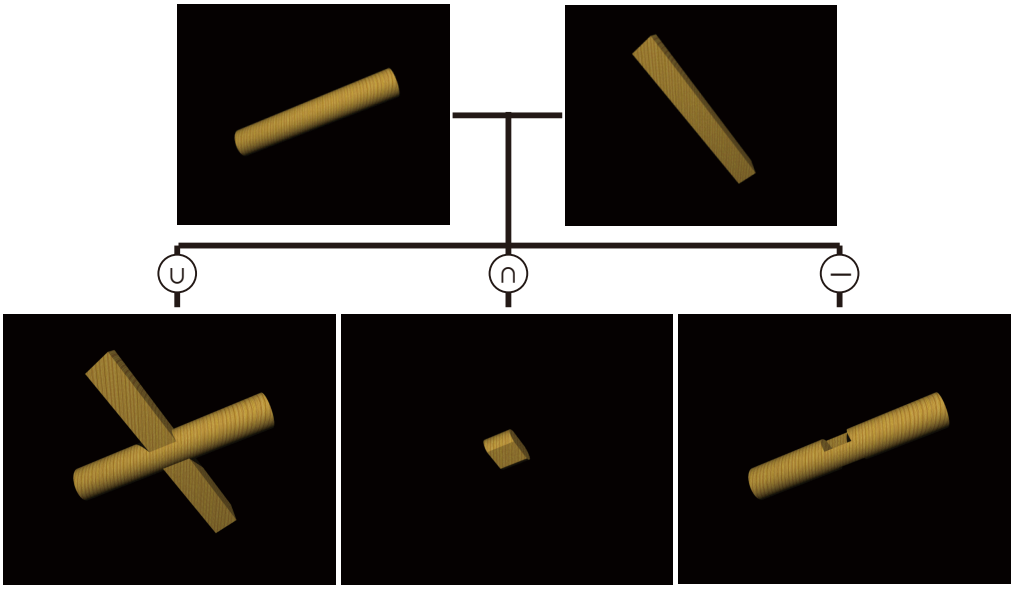


Fig. 1.5　CSG による表現．上段の 2 つの図形に対してブール演算を行うと下段のような形状が得られる．下段左から順に a) 和，b) 積，c) 差の演算を行った結果．レンダリング：Pov-Ray.

このうち，対象が固体もしくは液体の場合，複数の物体を扱うなど物体同士の衝突を考慮しなければならない場面が多く存在する．衝突判定を適切に処理しなければ，その生成されたアニメーションにおいて物体同士が貫通してしまう可能性があるため，CG における尤もらしい見た目を実現する際には欠かせない要素の一つである．衝突判定は，前節で述べた交差判定の一種とも考えることができる．多数の剛体を扱う場合や多数の要素で表されている弾性体や流体を扱う場合，物体の数を n とすると，すべてのペアについて衝突しているかどうかを判定する必要があるため，計算量は $O(n^2)$ となる．もちろん，物体の形状に依って，物体のペアについての衝突判定計算の負荷が異なり，球などの場合は比較的処理が軽いのに対し，ポリゴンで構成された物体は非常に負荷が高い．そこで，一般に前項で述べた分枝限定法の考え方に則り，broad phase と narrow phase の 2 つの処理に分けて衝突判定を行うことが多い．

broad phase で用いられるアルゴリズムには，形状近似や空間のインデクシング (indexing) などがある．形状近似による手法では，各物体を比較的簡単な形状で近似しておき，その形状での衝突判定を用いて，明らかに衝突しないペアを取り除く「枝刈り」処理を行う．一方，空間のインデクシングによる手法では，木などの階層構造や格子 (grid; グリッド) を用いて，空

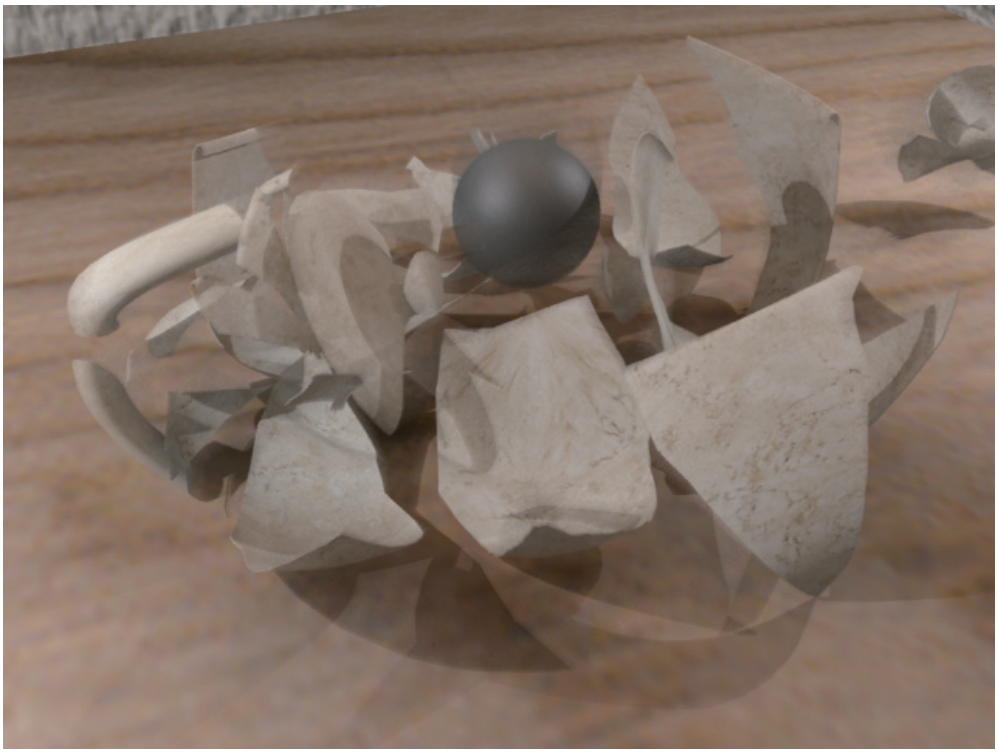


Fig. 1.6　物理ベース モデリングを用いた CG 表現．鉄球を上から落とし，ティー ポット モデルを破壊するシーン．破片同士で衝突判定が行われている．モデリング： 3ds Max，シミュレーション： RayFire Tool/PhysX，レンダリング: mental ray.

間探索を行い「枝刈り」する手法である．詳述は第 2 章に譲るが，uniform grid(一様格子) や BSP 木, BVH (Bounding Volume Hierarchy) を使ったアルゴリズムなどがある．また，sweep and prune というアルゴリズムも比較的よく用いられる [6].

一方，narrow phase では，貫通 (penetration) の防止やジョイントの処理，可動域の制限などを処理できるように，衝突するかどうかの情報以外にも，接触点の位置や交差領域の大きさ，法線などが厳密に計算される．特に CG では，複雑なポリゴン形状が用いられるため，その手法に関しても多く議論されている．凸形状に対する交差判定としては，凸性を利用して，極値を反復的に求める GJK アルゴリズム [32][*13]や前述の Voronoi 図を利用した Lin-Canny アルゴリズム [67, 68] などが提案されている．また，非凸形状に対しては DF (Distance Filed; 距離場) を用いた手法が提案されている [39]．さらに，ポリゴンが交差する部分の形状が必要な場合[*14]には，Muller ら [77] の手法を用いて，凸形状同士の交差形状を，凸包を利用することで

[*13] [32] の著者である E. G. Gilbert, D. W. Johnson, S. S. Keerthi 3 氏のイニシャルをとって GJK アルゴリズムとよばれている．

[*14] penalty 法とよばれる拘束解決の手法では，交差形状の体積が必要となる．

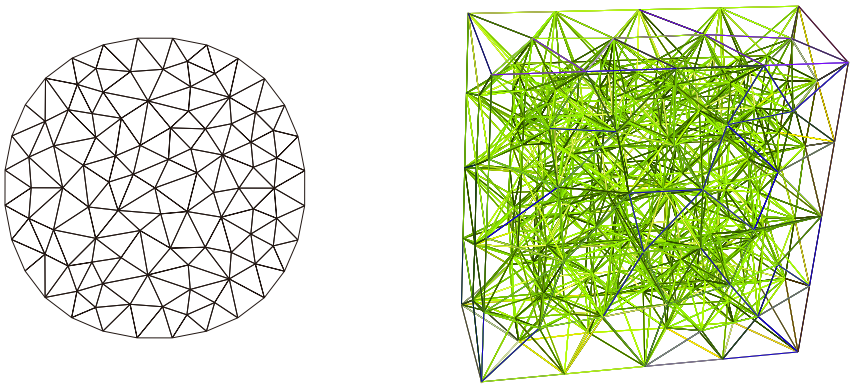


Fig. 1.7 Delaunay 三角形分割 (左) および四面体分割 (右) を利用して生成した FEM のための非構造メッシュの例.

求めることができる.

弾性体や布シミュレーション，キャラクタアニメーションのスキニングでは，同一物体内の異なるメッシュ同士の衝突判定に関しても考慮する必要がある．これは一般に self-collision とよばれる問題で，メッシュ同士での衝突判定を行う必要があるため，格子や BVH などの空間分割による枝刈りが行われる.

また，衝突判定以外でも物理ベース モデリング (physically-based modeling) では幾何アルゴリズムが使われる場面が多い．SPH (Smoothed Particle Dynamics)[76, 91, 79, 42] などの粒子法シミュレーションでは，KNNS や球と計算点の交差判定を用いて，近傍粒子を特定する処理が必要となる．一方，FEM や FDM (Finite Difference Method; 有限差分法)，FVM (Finite Volume Method; 有限体積法) といったメッシュを使って空間を離散化する手法では，メッシュの種類をある程度自由に選択でき，その質が計算の精度や効率にに大きく影響するため，形状に対してどうメッシュを切るか[*15]が重要になる．メッシュには構造メッシュ (structured mesh) と非構造メッシュ (unstructured mesh) が存在し，前者は立方体や正四面体などのまったく同じ形状が整列して並んだものを使うのに対し，非構造メッシュでは形状の違う六面体や四面体などを利用する．非構造メッシュの生成には前述の Delaunay 四面体分割を利用した手法 [31, 100] のほか，octree(八分木) を利用した手法 [116, 108]，advancing front 法 [72, 73] などが存在する．Delaunay 三角形分割および四面体分割を利用した非構造メッシュの生成例を (Fig.1.7) に示す.

[*15] この作業をメッシング (meshing) とよぶ.

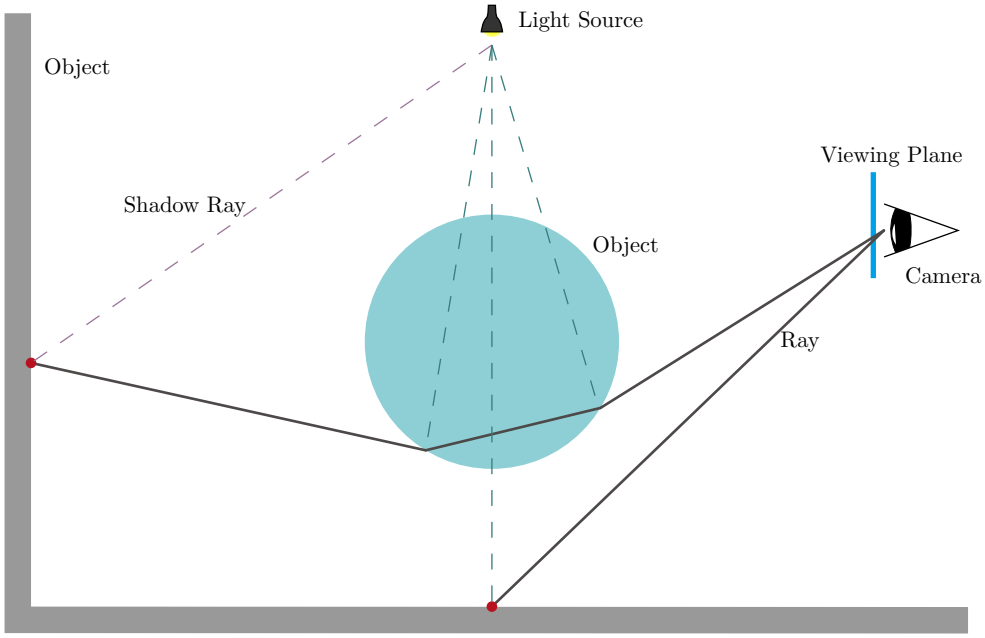


Fig. 1.8　レイ トレーシング法の模式図．影かどうかの判断については，それぞれの点から shadow ray とよばれるレイを光源に向かって飛ばすことで判断する (点光源 (point light source) の場合).

空間表現とレンダリング

空間上にモデリングされた形状は，レンダリング (rendering) によって，画像として出力される．レンダリングの際には，見えない面や線を取り除く陰面消去とよばれる処理が行われる．

CG の黎明期には，Z ソート法 (z-sorting, painter's algorithm; 画家のアルゴリズム) を用いて陰面消去が行われていた．物体の面を視点から遠いものから順番にソートし，その順に描画していくという手法である．その後，Z バッファ (z-buffer) 法が提案されたが，当時のハードウェアに対して計算コストが高価であった．そこで Fuchs ら [28] らは BSP 木を利用した手法を提案し，その後 Gordon ら [36] は高速なレンダリング手法として応用した[*16]．現在は，ハードウェアの進化とともにリアルタイム用途では Z バッファ法が主に用いられている[*17]．

一方，プリレンダー (非リアルタイム) 用途では，主にスキャンライン (scanline) 法が使用されていたが，1980 年以降のレイ トレーシング (ray tracing; 光線追跡) 法の発展により，

[*16] 1993 年に発売された DOOM というゲームには，この BSP 木を使ったレンダリング手法が実装され，当時としては画期的なグラフィックスのクオリティを実現していた．

[*17] 一部のゲーム機等，メモリの制約がある場合にはスキャンライン法が用いられている場合もある．

多くのレンダラーでレイトレーシング法が使用されるようになった．レイ トレーシング法は Appel[2] によるレイ キャスティング (ray casting; 光線投射) 法に端を発した[*18]．その後，Goldstein ら [34] により二次曲面の描画が，Kay ら [55] により透明物体の描画が実現され，Whitted ら [112] により反射，屈折，光源からの影が正確に反映された再帰レイ トレーシング (recursive ray tracing) 法が提案された．そのアルゴリズムは，視点からスクリーン (viewing plane) に向かって光線 (ray) を飛ばして直進させ，最初に物体とぶつかった地点を探す．その地点で反射・屈折を行い，その方向に新たな光線を生成する．これを繰り返し，最終的に光線が追跡したすべての情報を視点に集め，スクリーンに投影することで画像を生成するというものである．模式図を (Fig.1.8) に示す．この際，光線と物体の交差判定を行うことで，「最初に物体とぶつかった地点」を計算することになる．これは線分と primitive[*19]の交差判定に帰着する．現在，一般に実用とされているシーンでは数万〜数十億 primitive 程度，光線の数も数十万〜数十億 ray 程度と問題の大規模化が進んでいる．これらそれぞれに対し，ひと組ずつ交差判定を行うことは現実的ではないことは明らかである．そこで，一般に分枝限定法を利用した高速化・効率化 (acceleration) が行われる．主に利用される空間データ構造として，uniform grid，octree，kd-tree (K-Dimentional Tree; kd 木)，BVH，およびその発展手法などがあげられる．

その後，レイ トレーシング法に関しては，分散レイ トレーシング (distributed ray tracing) 法 [16]，モンテカルロ パス トレーシング (Monte-Carlo path tracing, MCPT) 法 [52]，双方向パス トレーシング (bidirectional path tracing, BDPT) 法 [59, 106]，MLT (Metropolis Light Transport; メトロポリス光輸送)[107] などの発展的手法が提案されている (Fig.1.9)[*20]．MCPT, BDPT, MLT 各アルゴリズムの概略図を (Fig.1.10) に掲載する．これらの手法により，光源からの直接光によるライティング効果だけでなく，周囲の物体に反射して照らされるグローバル イルミネーション (Global Illumination, GI; 大域照明，広域照明) の効果を得ることができ，より自然で写実的な映像表現が可能となった．しかし，レイ トレーシング法でグローバル イルミネーションの効果を得るためには，さらに大量の交差判定が課せられるため，計算速度向上のため

[*18] レイ キャスティング法はレイ トレーシング法の前身ともいうべき存在で，非常によく似ているが，レイ トレーシング法では，レイが反射・屈折を繰り返しながらシーン情報を集めるのに対し，レイ キャスティング法は当初，視点から一番近い面，すなわち可視面を見つけるために考案されたアルゴリズムである．その後，レイ キャスティング法は炎や煙などのボリューム データを可視化する手法として発展した [17, 115].

[*19] ここでは，シーンを構成する球や 2 次曲面，ポリゴンの三角形などを指す.

[*20] これらの手法はモンテカルロ サンプリングの理論と密接にかかわっている．近年でも，Cline らの ERPT (Energy Redistribution Path Tracing; エネルギー再分布経路追跡) 法 [14] や PMC-ER (Population Monte-Carlo Energy Redistribution; ポピュレーション モンテカルロ エネルギー再分布) 法 [60] などの派生的手法が多く研究されている.

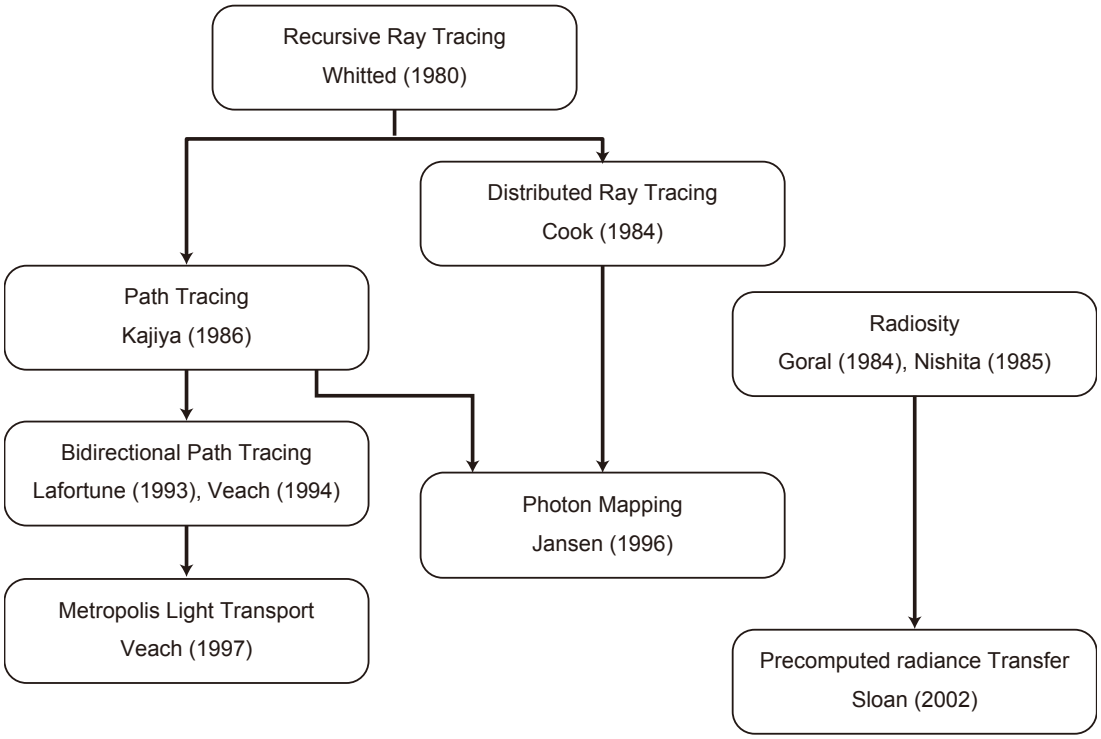


Fig. 1.9　代表的なレンダリング手法とその流れ.

の空間データ構造 (acceleration structure) の質がより重要になる.

一方で，Goral ら [35] や Nishita ら [82] によって，ラジオシティ (radiosity) 法とよばれる，グローバル イルミネーションを実現する手法が提案された [15]．ラジオシティ法では，シーン中の物体をパッチとよばれる微小要素に分割し，閉じた空間における光エネルギーの平衡状態を求めることで輝度を決定する．ラジオシティ法を用いることで，レイ トレーシングに比べ影のエッジが立たないなど，比較的輝度分布がゆるやかな，やわらかい映像表現が可能となるため，建築設計などの現場でよく用いられている．初期のラジオシティ法ではすべてのパッチについて連立一次方程式を解くという，非常に計算コストの高いアルゴリズムをとっていたが，その後，各三角形メッシュから次々とエネルギーを放射していき，未放射エネルギーの和が一定以下に収束するまで反復を繰り返すプログレッシブ リファインメント ラジオシティ (progressive refinement radiosity) 法が提案され，広く用いられている．この手法では，特にメッシュの質が生成される映像の品質と計算速度を左右するため，前述の物理シミュレーションでのメッシュ同様，空間表現が重要な要素となる．この際，discontinuity meshing[44]

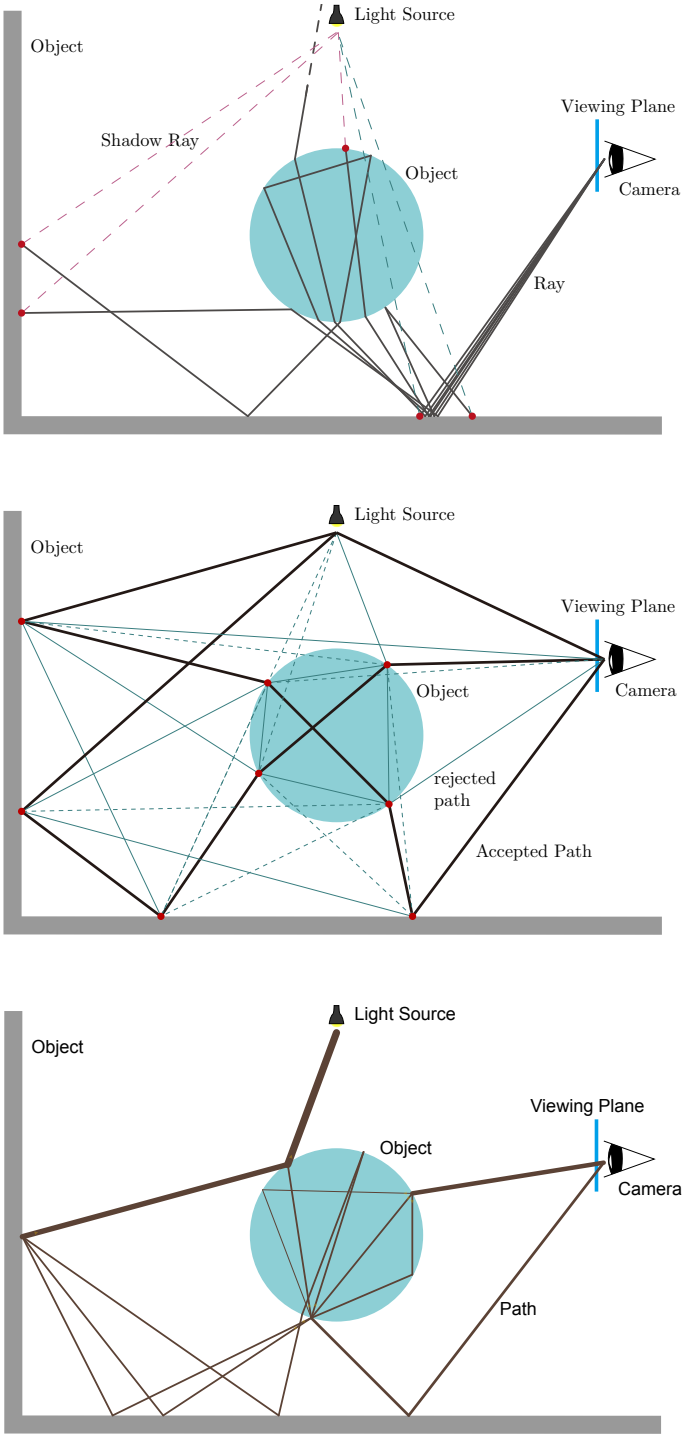


Fig. 1.10　MCPT, BDPT, MLT の模式図．上から順に，a) MCPT では，光線が物体にあたると，その点から確率的に反射・屈折を選択 (モンテカルロサンプリング) し，新たな光線を生成する．b) BDPT では，光源と視点の両方から光線を生成し，物体との交点同士について放射輝度の大きさを推定することでパスを選択し，サンプリングを行う．c) MLT では，視点からの光線経路に対し戦略を立てながら追跡を行っていく．

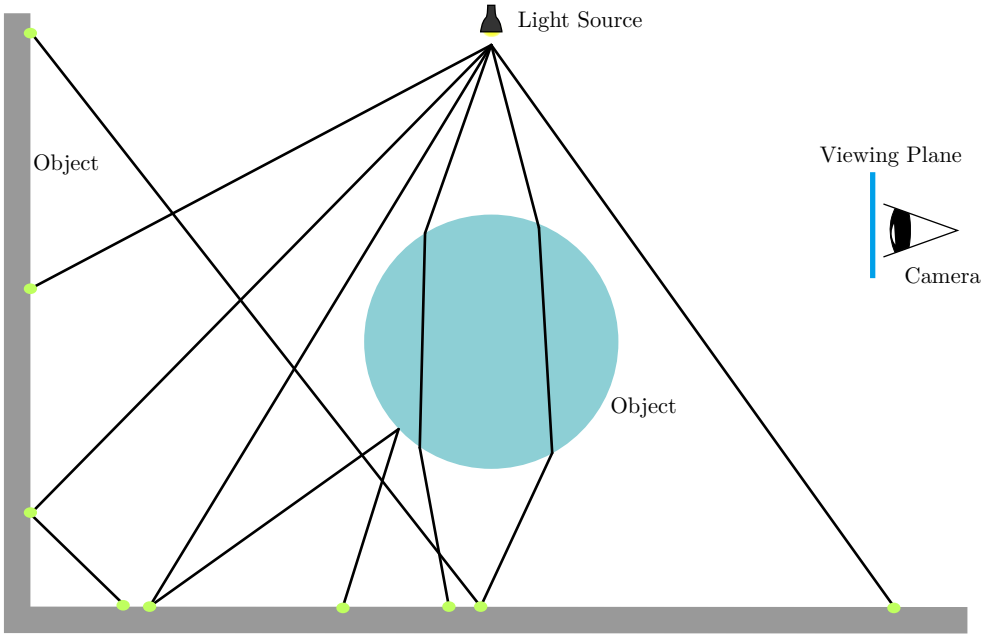


Fig. 1.11　フォトン マッピング法の 1pass 目の概略図.

などの adaptive なメッシング手法が比較的よく用いられる．ラジオシティ法からの派生として，PRT (Precomputed Radiance Transfer; 事前計算放射輝度伝播)[101] があり，放射輝度の伝播を事前計算し，球面調和展開しておくことで，その係数を利用してリアルタイムにグローバルイルミネーションを計算することができる．動的な形状の変形に関しては，Ren ら [92] が形状を球体近似することで PRT を実現する手法を提案している．

また，フォトン マッピング (photon mapping) 法 [50] や MLT (Metropolis light transport; メトロポリス光輸送) もグローバル イルミネーションを実現する手法としてあげられる．前者はフォトンを光源から大量にシーン中に飛ばし，レイ トレーシング法によりそのフォトンを集める (final gathering) という 2 つのパス (pass) を経てグローバル イルミネーションを実現している (Fig.1.11)．これにより，水などの屈折物体や金属のような反射物体に対し，コースティクス (集光模様，火線) を表現することができる．なお，使われるフォトン数は数万〜数十億に上る．レイ トレーシング法と同様の直線と primitive との交差判定のほかに，2pass 目でのフォトン収集において KNNS などを用いた近傍フォトン探索のアルゴリズムが必要となり，フォトン マップにおいても acceleration structure の重要度は高い[*21]．フォトン マッピング法

[*21] なお, フォトン マッピング法については, Herzog ら [46] の photon ray splatting, Hachisuka ら [40] の progressive photon mapping などの発展的手法が提案されている．

を用いたレンダリング画像の比較を (Fig.1.12) に掲載する．

以上で述べてきたレンダリング手法は，主にメッシュ構造からなる primitive を描画する手法である．そのほかにも，NURBS 曲面などのパラメトリック曲面，メタボール (metaball, blob) などの陰曲面，participating media などの volume data といった特殊な材質形状や point-based rendering と言った特殊なレンダリング手法を用いることで，リアリティの向上や表現性の向上を図ることがある．これらのレンダリングには，通常とは異なる空間表現が使用されることが多い．メタボールでは等値面の描画に関して，メッシュを生成する方法のほかに，uniform grid などの空間構造を用い，ボクセル (voxel) に対してマーチング キューブ (marching cube) 法を適用したり，レイ マーチング (ray marching) とよばれる再帰的探索手法を用いたりすることで実現することができる．また，point-based rendering の手法として，QSplat[96] や SPT (Sequantial Point Tree) [18] とよばれる階層的構造をもったデータ構造が提案されており，大規模データに対する LOD (Level Of Detail) が可能となった．

その他の応用

モデリングやレンダリング以外の分野でも，空間データ構造をうまく応用したシステムが提案されている．

Furukawa ら [30] は，写真から 3 次元の空間データを再構築する手法を提案した．Manhattan-World 仮説[*22]を仮定し，uniform grid を使って，軸並行 (axis-aligned) な深度マップ (depth map) を構築し，Delaunay 三角形分割により 3 次元のメッシュモデルとして展開することで実現している．

1.2.3 リアルタイムアプリケーションに向けて

CG 分野では，特にリアルタイム化が課題となることが多い．近年の，ゲームへの積極的な応用やメディア アート (media art) および AR[*23]の一般への広がりから，高品質な映像をインタラクティブに実現する要求が高まってきている．一般に，リアルタイム CG では秒間 30 フレームもしくは 60 フレーム程度が理想とされており，課せられる計算時間は 15〜33 ミリ秒

[*22] アメリカ合衆国ニューヨーク市のマンハッタンの通りが規則的な格子状に並んでいることに由来し，このような碁盤目状の都市で，2 地点の通りに沿った最短経路の長さをマンハッタン距離という．Manhattan-World 仮説は，都市や人工空間ではその輝度分布に直交性が強く表れるという仮説であり，これから壁や障害物の位置を推定する．

[*23] iPhone や Android 携帯など，ユビキタス端末の普及やデジタル サイネージへの期待から，セカイカメラ (頓智・) やテノヒラアド (博報堂) といった実際のビジネス応用も広がりをみせている．

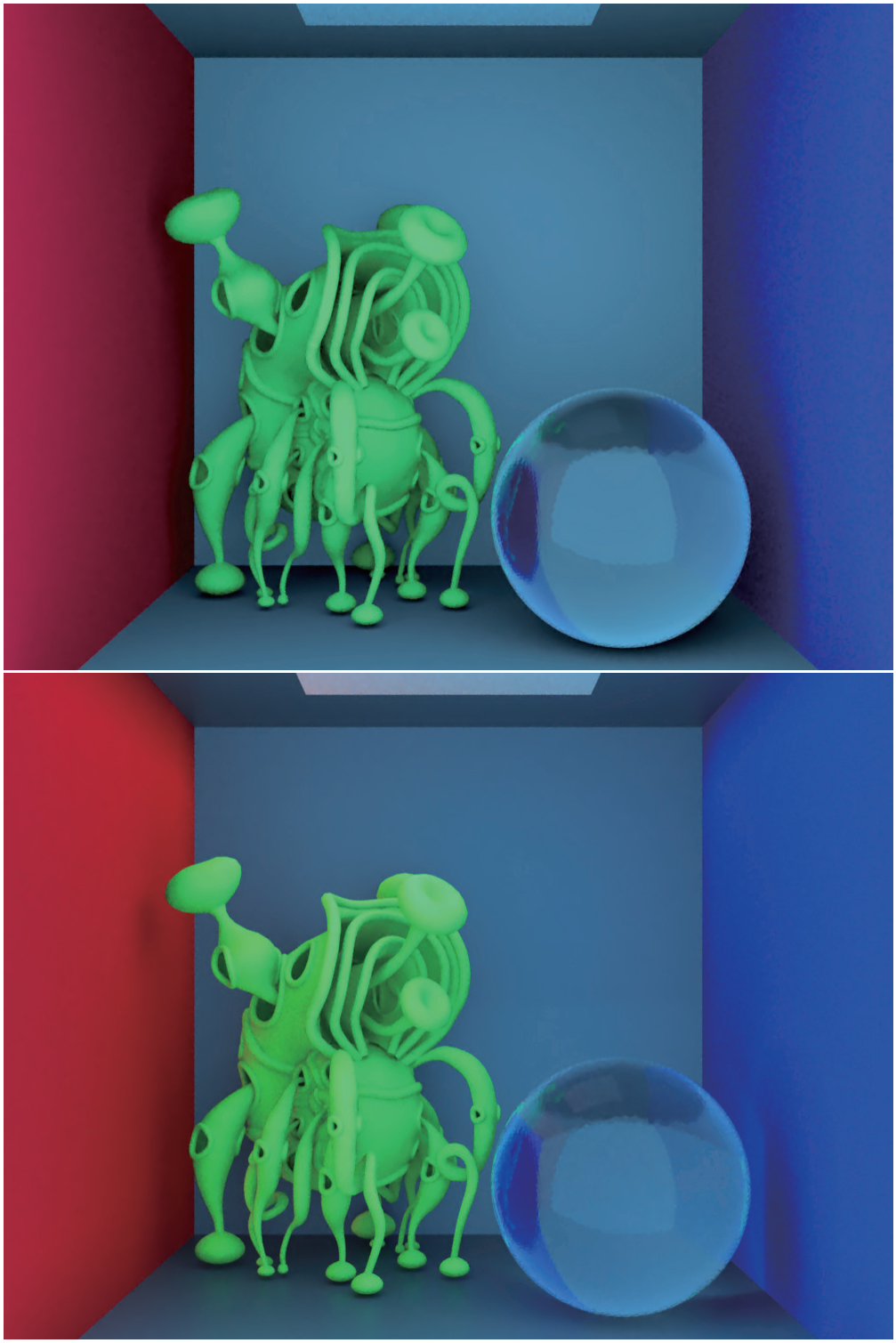


Fig. 1.12　レイ トレーシング法 (上図) とフォトン マッピング法 (下図) のレンダリング画像の比較．フォトン マッピング法では，モデルの左部への赤色の映り込みや，床面への緑色の映り込み，球の周りへの配光などが確認できる．モデルデータの著作権は河口洋一郎に帰属する．モデリング: Blender，レンダリング： LuxRender.

である．近年のハードウェアの目覚ましい進化により，短い計算時間の間にリアルタイムで解ける問題のスケールは日々大きくなり，新たな応用も徐々に出現し始めている．しかし，リアルタイム化に際しては，既存のアルゴリズムをそのまま用いることができる場合は極めて少なく，多くの場合，リアルタイム専用の最適化ないし簡略化を行わなくてはならない．これは，すでに述べたように計算幾何アルゴリズムの多くが多項式オーダーの計算量であることに起因するが，もう一つの背景に，現在のプロセッサにおける技術的傾向がある．

2004 年ごろから発熱や消費電力の影響で，CPU のクロック周波数が 4GHz 程度で頭打ちになったことから，各社ともマルチ コア化およびハードウェア マルチスレッディング (hardware multi-threading) へと戦略をシフトした．2010 年現在，Core i7 (Intel) では 4 コア搭載，同時マルチスレッディング技術により最大 8 スレッドが同時実行可能，Opteron (AMD) では 6 コア搭載，UltraSPARC T2 Plus (富士通，サン) では 8 コア搭載，最大同時 32 スレッドとなっており，今後もさらなるマルチ コア化が予想されている．また，グラフィックス専用のハードウェア ユニットである GPU では，CPU より以前から高度にコア並列化されたアーキテクチャ設計を採用しており，現在では最大 240 コア，ピーク性能 1TFPS に及ぶものも存在する．これらの，同種 (homogeneous) のコアが複数搭載されている演算ユニットの他に，異なった種類 (heterogeneous) のコアが複数搭載されているユニットも開発されている．PlayStation 3 (SCE) にも搭載されている Cell Broadband Engine (Cell BE) は，1 基の PPU と 8 基の SPU で構成されている．

このようにコア単位での並列化が進むにつれ，そのアルゴリズムも並列実行に適したものにする必要がある．並列化にはタスク並列 (task parallelism) とデータ並列 (data pararellism) という 2 つのアプローチが存在する．タスク並列では，問題を比較的大きな複数のタスクに分割し，各スレッドにタスクを割り当て，同時に異なる種類のタスクを処理させる考え方である．一方，データ並列の考え方は，大きなデータに対して同じ処理を一度に行うことで並列化を図るというものである．GPU は，データ並列に最適化することで高速化を図っている演算ユニットである．

1.3 目的

本章で述べてきたように，空間データ構造は非常に多くのアルゴリズムやシステムで利用されており，その品質および効率的な構築がシステム全体のパフォーマンスに影響を与えるといえる．空間探索のための空間分割手法においては，空間の局在性を考慮に入れることが高品質化の鍵となる．また，リアルタイム化への要求から，データ パラレルを意識した並列アルゴリ

ズムの考案が重要である．これらの背景をうけ，本研究では空間探索のための新たなデータ構造の提案を目的とし，空間の局在性を考慮にいれた高品質で，並列計算に適した構築モデルを提案する．

第 2 章

関連研究

2.1 空間分割データ構造に関する研究

空間分割データ構造は分枝限定により効率的に空間を探索するためのデータ構造である．この空間分割データ構造は探索に必要なデータ構造を構築するの必要な計算時間とメモリ量および空間探索の問合せ (query) に応答するまでの処理時間によって評価することができる．

2.1.1 uniform grid

もっとも単純な空間分割の方法は，空間を均一に同じ図形で区切ることである．これは，uniform grid (regular grid) とよばれるデータ構造である．空間探索の場合, 主に axis-aligned(軸平行) な領域で分割され，一つ一つの領域は voxel と呼ばれる．各 voxel はその voxel に存在する要素へのポインタやインデックスを所持している．要素数が n の場合構築にかかる時間は $O(n)$ で，非常に少ない計算で構築可能な点が特徴である．その反面，品質が悪い点が欠点としてあげられ，物体の分布が一様でない場合には極端にその効率が低下し，さらに余分なメモリを大量に消費してしまう．例えば，広い競技場の中央にぽつんとティーポットが置かれているような状況では，なにもない空間のために大量の voxel を確保し，大部分の primitive が数個の voxel にまとめて格納されてしまい，パフォーマンスの低下と余分なメモリの消費を引き起こしてしまう．このような問題は，俗に”teapot in a stadium”問題と呼ばれている．uniform grid の概略図を (Fig.2.1) に示す.

uniform grid は古くから使用されており，Levinthal[66] はコンピュータによる分子構造の解析に 3D cubing という名前で使用した．これは，原子同士の相互作用を計算する際に，近傍探索を行い不要な要素を排除するためのデータ構造であった．Fujimoto ら [29] は初めて

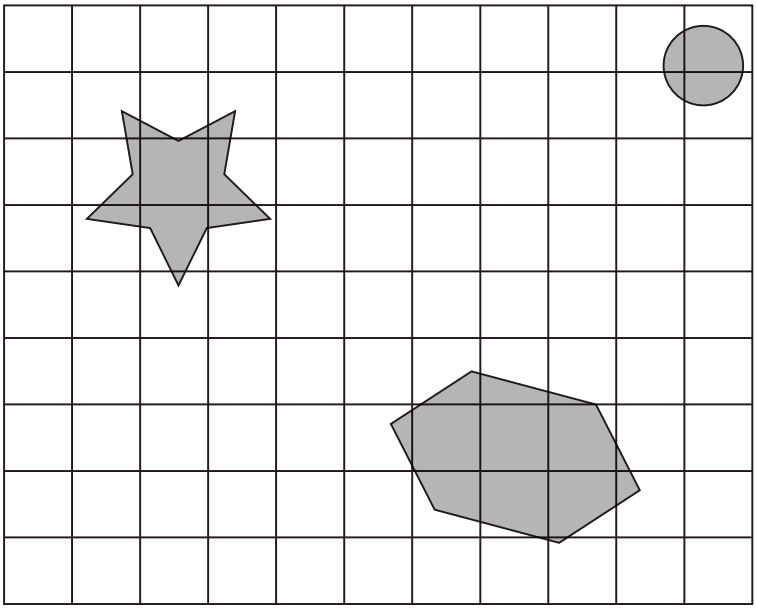


Fig. 2.1　uniform grid の概略図．uniform grid は空間を均一な大きさの cell に分割する．各 cell はオーバーラップする物体への参照を保持する．

uniform grid をレイ トレーシング法に導入した．光線との交差判定に関しては，3D DDA (3-dimensional Digital Differential Analyzer; 3 次元デジタル微分解析器) を用いた．2 次元での DDA 法の概要を (Fig.2.2) に示す．プログラマブル シェーダによる GPU 実装のレイ トレーシングでは，その性質上 uniform grid が使われることが多く，Purcell ら [89] はプログラマブル シェーダを用いて，テクスチャ メモリ上に uniform grid を生成し，Whitted レイ トレーシングやパス トレーシングへ応用した．また，Purcell ら [90] はフォトン マッピング法についても GPU での実装を行い，近傍フォトン探索に uniform grid を用いた．

uniform grid の不要なメモリを多く使用するという欠点を補うために，hash を使った hash grid が提案されている [114]．Teschner ら [103] は，self-collision 解決のために 3 次元での hash grid を用いた手法を提案している．また，Harada ら [41] は，同様の目的でスライスグリッドという空間データ構造を提案し，SPH による流体シミュレーションへの応用を行った．これは，uniform grid をある軸に着目した場合に，データが存在する voxel の下限と上限の間のみをメモリ上に確保することで，それ以外の領域に必要なメモリを節約し，キャッシュのヒット率をあげる効果がある．

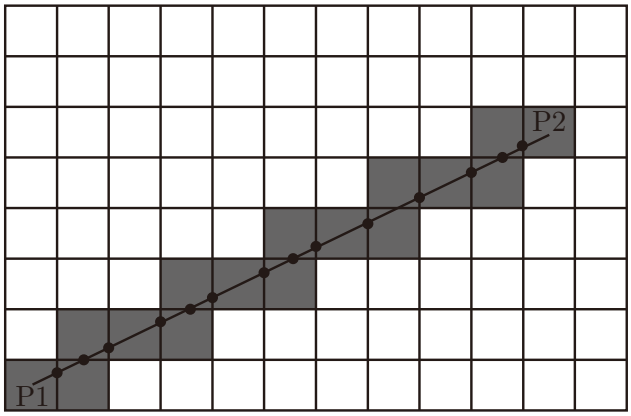


Fig. 2.2　DDA 法の 2 次元での例．点 P1 と点 P2 を結ぶ光線に対して，交差する voxel を求めるときは，P1 からスタートして，次に voxel の壁と光線がぶつかる点をそれぞれの軸について計算し，一番早く光線がぶつかる点へ進む．光線の終端 P2 にたどり着くまでこれを繰り返す．

2.1.2　octree

uniform grid における，対象の空間的局在性を考慮に入れていないという点を改善する標準的な手段の一つが，octree/quadtree である．前者は 3 次元の場合でリーフ以外の各ノードが 8 つの子を持つ木構造，後者は 2 次元の場合でリーフ以外の各ノードが 4 つの子を持つ木構造である．空間を 8/4 つの均等な大きさの領域に再帰分割することでトップダウン的に構築することができる．quadtree の例を (Fig.2.3) に示す．

一般に n 個のリーフが存在する深さ d の octree は，$O((d+1)n)$ のノードをもち，$O((d+1)n)$ の時間で構築できる．また，探索に関しては，$O(\log n)$ の時間で行うことができる．octree は実装の容易さから古くからさまざまな応用例が存在する．

Bandi ら [5] は衝突判定に，Hernquist ら [45] は SPH の近傍探索に Jevans ら [51] および Revelles ら [93] はレイ トレーシング法への応用を行った．また，テクスチャ座標を用いずに octree の階層構造が色データを持つことによって，テクスチャを表現する octree texture というデータ構造が提案され [9]，GPU 上への実装も行われている [64].

また，レイ トレーシング法では 5D octree という空間探索手法が提案されている [3, 4]．こ

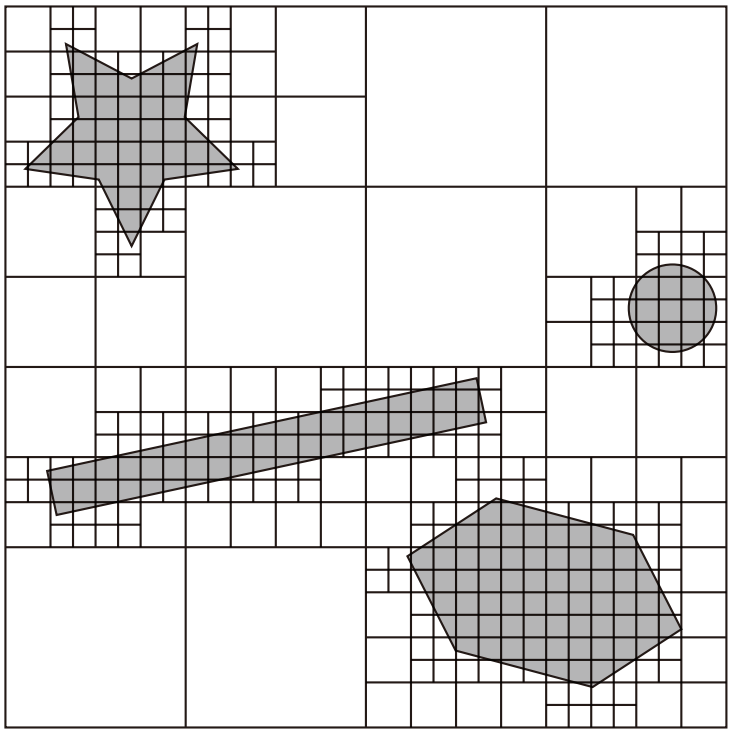


Fig. 2.3　quadtree の概略図．quadtree は再帰的に空間を 4 分割する手法．図では最大深度が 4 となっている．

れは，レイ トレーシングにおける光線の端点 $\boldsymbol{o} \in O = \mathbb{R}^3$ と方向を立方体にマッピングした情報 $\boldsymbol{p} = (u, v) \in P = [-1, 1]^2$ によってあらわされた 5 次元空間 $O \times P$ で光線を離散化し octree を beam で探索することにより，効率化を図ろうというものである (Fig.2.4).

2.1.3　BSP 木と kd-tree

octree では再帰的に空間を 8 分割していくが，BSP 木では再帰的に空間を 2 分割していく．オブジェクトの集合 S に対して，$S(\nu)$ をノード ν に関連付けたオブジェクト集合とする．このとき，BSP $T(S)$ は次のように定義される [61].

1. $|S| \leq 1$ ならば，T は $S(\nu) := S$ を格納するリーフ ν
2. $|S| > 1$ ならば，T のルートはノード ν で，ν は平面 h_ν と集合 $S(\nu) := \{x \in S | x \subseteq h_\nu\}$ を格納する．ν は 2 つの子 T^-と T^+ ももち，それぞれ $S^- := \{x \cap h_\nu^- | x \in S\}$ および

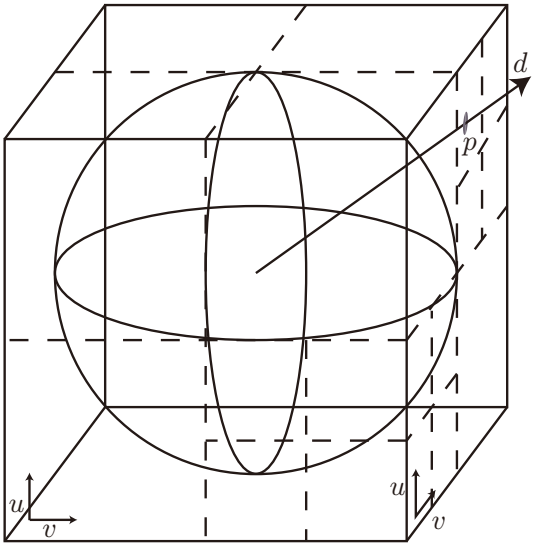


Fig. 2.4　5D octree の模式図．単位球の外接立方体の 6 つの面それぞれに $(u, v) \in [-1, 1]^2$ の座標系を考える．光線の端点 o と (u, v) であらわされる 5D octree を 6 つ構築することで，5D octree によって，各光線では 3 次元上でビームの定義に分類され，ノードを構築することになる．これによって，各光線のコヒーレンス (coherence) を利用することができ，効率的に空間を探索することができる．

$S^+ := \{x \cap h_v^+ | x \in S\}$ をあらわす BSP である．

なお，h は $\mathbb{R}^d$ 中の平面，h^+ および h^- はそれぞれ正と負の半空間とする．(Fig.2.5) に概略図を示す．

ポリゴンをオブジェクトの対象とし，分割平面としてポリゴンを含む空間が選ばれるとき，再帰的な分割により，構築には $O(n \log n)$ の時間，query には $O(\log n)$ の時間がかかり，使用するメモリの量は $O(n)$ となる [20, 85].

そのほかにも，BSP 木の構築には様々な手段をとることができる．その代表的な特殊化が kd-tree であり，これは分割平面を axis-aligned な平面に限った BSP 木である．直線との交差判定を行う場合，軸垂直な平面と直線との交差判定のみでよく，計算負荷が軽いため，レイ トレーシング法では非常によく利用されている．ただし，基本的にトップ ダウンで構築されるため，構築が遅いという欠点がある．

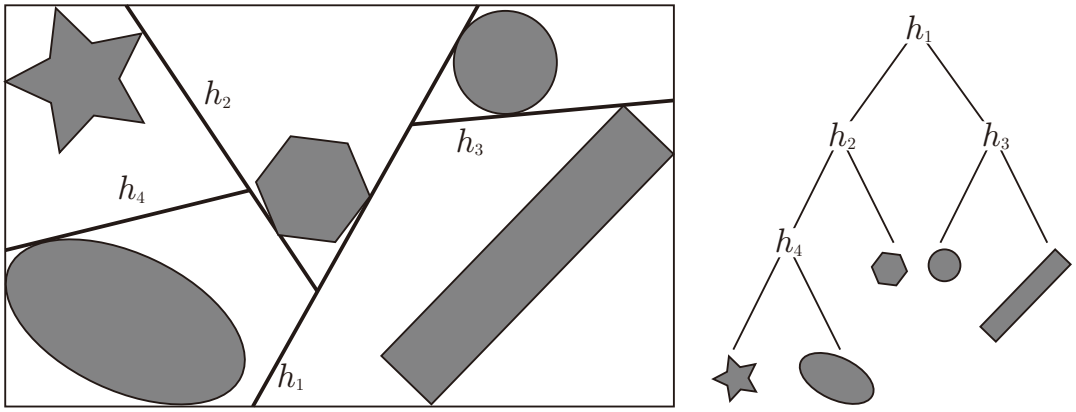


Fig. 2.5　BSP の概略図と BPS 木．オブジェクトに対する BSP の例．リーフ ノードは空間を表し，それ以外のノードは分割平面を表す．

kd-tree の場合，どのような戦略で分割平面を選ぶか (分割スキーム) が木構造の質を左右し，ひいてはアプリケーションにおける効率性を決定する．主な戦略として，object median, spatial median およびコスト関数を利用した方法があげられる．

object median では，分割すべき空間に存在するオブジェクト n 個をソートしたときに，頭から $n/2$ 個目の位置に存在するオブジェクトの位置で分割平面を定義する方法である．このとき，ソートの方向，すなわち分割の方向は分割すべき空間の最長辺を使うと効率的である [22]．object median による kd-tree の構築例を (Fig.2.6) に掲載する．この場合，深さ $O(\log n)$ のバランスのとれた平衡木が生成される．

一方，spatial median では，分割すべき空間をその最長辺の中点を通る平面で分割する手法である．この場合は平衡木にはならないが，内部のオブジェクトをソートする必要ないため構築が比較的早い．コスト関数を使った構築法では，レイ トレーシング法では SAH (Surface Area Heuristic)，衝突判定やフォトン マップ法では VVH (Voxel Volume Heuristic) などが有名である．そのほか，sliding midpoint split[75] など様々な分割手法が提案されている．(Fig.2.7) にその一例を示す．

kd-tree は Bentley[10] によって初めて考案された．挿入や削除の手法，NNS や範囲探索などの探索手法のほか，kd-tree の最適化などを提案した．Kaplan ら [54] は kd-tree を初めてレイ トレーシング法へ応用した．Kaplan らの構築手法は spatial median による分割手法であった．その後，Jansen ら [48] によって，効率的な traversal(走査) のアルゴリズムが考案された．

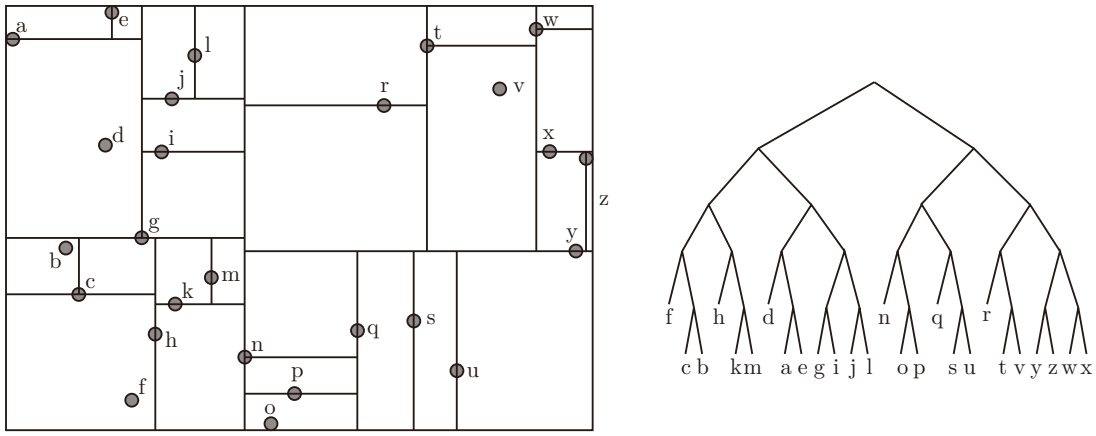


Fig. 2.6　object median 戦略による kd-tree の構築．平衡木が生成されていることが分かる．

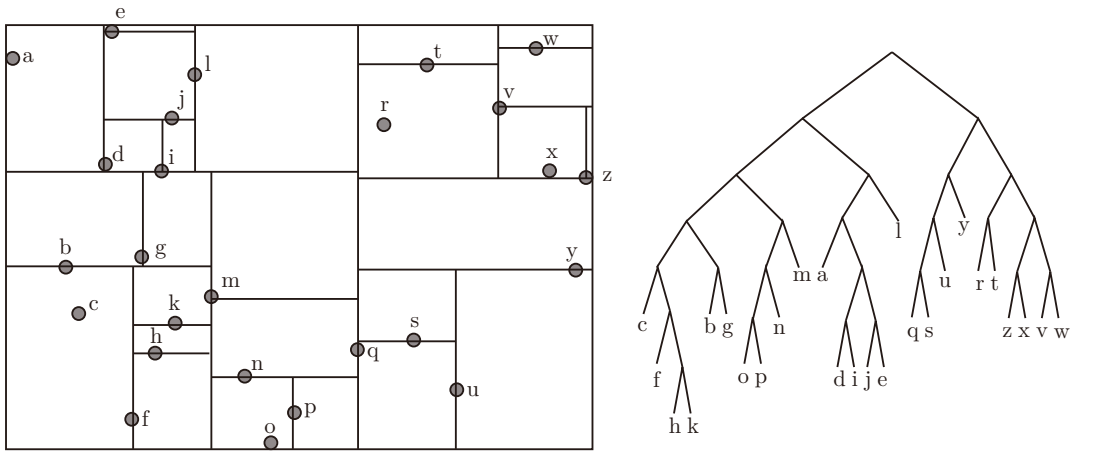


Fig. 2.7　改変 sliding midpoint split による kd-tree の構築．空間の中点から正の方向に進んだときに，初めて存在するオブジェクトの位置で分割している．分割後の空間の片方が空になる場合は，少なくとも 1 つはオブジェクトが存在するように分割平面を選択する．object median と同じデータに対する分割であるが，平衡木となっていないことが確認できる．

MacDonald ら [74] は，SAH による分割手法を開発した．コスト関数 C は以下のように定義されている．

$$C = \frac{C_i \sum_{i=1}^{N_i} SA(i) + C_l \sum_{l=1}^{N_l} SA(l) + C_o \sum_{l=1}^{N_l} SA(l) \cdot N(l)}{SA(root)} \tag{2.1}$$

ここで，C_i，C_l，C_o はそれぞれ，内部ノード (interior node)[*1]の traversal コスト，リーフの traversal コスト，オブジェクトと光線との厳密な交差判定にかかるコストを，N_i は内部ノードの数，N_l はリーフの個数，$N(l)$ はリーフ中のオブジェクトの数，$SA(i)$ は内部ノード i の表面積，$SA(l)$ はリーフノード l の表面積を表す．

また，Bentley ら [11] は準動的な点集合に対して，動的にデータを構築する手法を提案した．

近年の GPU でのプログラマビリティの向上により，GPU レイ トレーシングで空間データ構造に kd-tree を採用する場合も増えてきている．Foley ら [25] は，CPU で構築した kd-tree を GPU でのレイ トレーシング法に応用した．Foley らの手法では，traversal の際に stack を使わない点が特徴である．Popov ら [87] は，packet tracing という複数の ray を効率的に処理する手法を提案し，GPU 上での実装を行った．Zhou ら [120] は GPU での kd-tree 構築を行い，フォトン マップ法などの実装を行った．

一方で，CPU における kd-tree の並列化に関する研究も行われた．Shevtsov ら [99] は，CPU における実装で，kd-tree の構築を効率化した．複数の core における並列化により，2,00,000 ポリゴン程度の動的な影，テクスチャ付きシーンで 1024x1024 の解像度において，7-12 fps (Frame Per Second) の速度を達成した．

さらに，専用のレイ トレーシング用のハードウェアを設計し，kd-tree を実装したものも存在する．Woop ら [113] は，そのハードウェアに RPU (Ray Processing Unit) と名付け，FPGA (Field Programmable Gate Array) にそのプロトタイプを実装した。

2.1.4　BV と BVH

複雑な形状に対する幾何アルゴリズムに対して，複雑な形状を覆う簡単な形状で置き換えてその適応を行い，それをパスしたもののみ複雑な形状で厳密なテストをすることで，計算負荷を抑える方法が BV を用いた手法である．3 次元の BV の例として，bounding sphere, AABB (Axis-Aligned Bounding Box), OBB (Oriented Bounding Box), LSS (Line Swept Sphere,

[*1] リーフ以外のノードを表す.

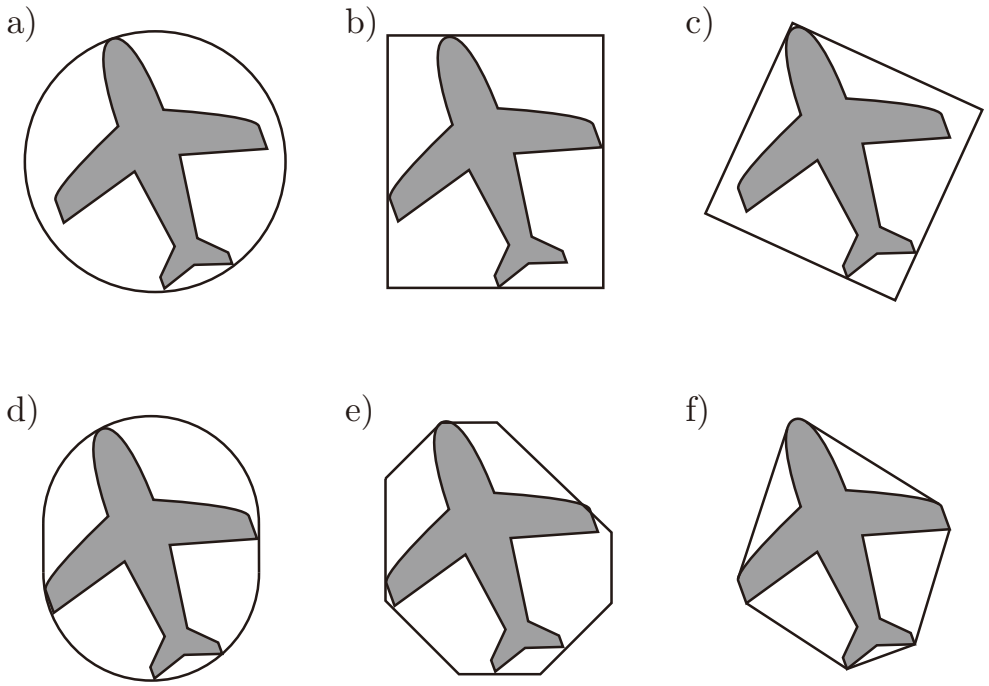


Fig. 2.8　BV の例 (2 次元)．左上から a) bounding sphere，b) AABB，c) OBB，d) LSS，e) DOP，f) 凸包

capsule)，角柱，円筒，k-DOP (k-Discrete Oriented Polytope)，凸包などがある．それぞれの例は (Fig.2.8)，(Fig.2.9) を参照されたい．

さらにこれらを，階層的にし，木構造を作ったものが BVH[*2]である．BVH は BV の形状によって sphere tree[47, 84]，AABB tree[118, 105, 62]，OBB tree[4, 37]，DOP tree[56, 58, 117]，convex hull tree[24] などに分類できる．構築方法にはトップダウンによる手法，ボトムアップによる手法, 挿入による手法がある．kd-tree 同様, spatial median や object median, SAH, VVH などの分割スキームが存在する．2 次元での BVH のトップダウンでの構築例を (Fig.2.10) に示す．

BVH は，衝突判定やレイ トレーシング法で多く採用されている空間データ構造である．Rubin ら [95] は，初めて BVH をレイ トレーシング法に応用した．Rubin らの手法では，分枝

[*2] 2 次元では volume は存在しないため，一般に BVH とはよばない．長方形による階層構造は R tree とよばれ，構成長方形は MBR (Minimum Bounding Rectangle) とよばれる．なお，R は rectangle の略である．さらに R* tree[8] や R+ tree[98]，Hilbert R tree[53] といった派生的手法も研究されている．

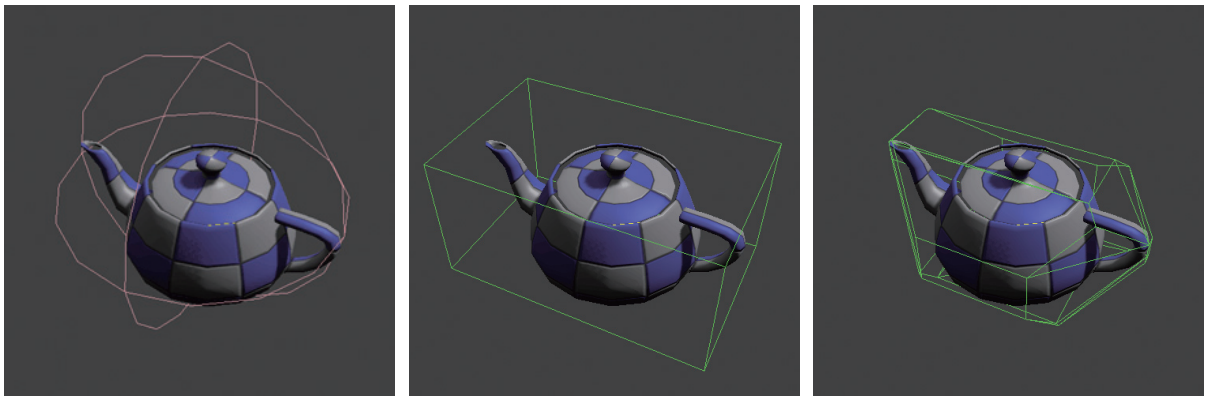


Fig. 2.9　BV の例 (3 次元)．左から a) bounding sphere，b) AABB，c) 18-DOP．k-DOP は k 個の slab の交差によってあらわされているる．18-DOP は 18 個のスラブ

限定法のような枝刈りの目的ではなく，形状のコンテナとしての利用であった．Roussopoulos ら [94] らは，画像データベースにおける空間探索に R tree を使用した．この手法では各点から最近接点を探し，その距離でソートすることにより，ボトムアップで構築している．Goldsmith ら [33] は自己構築型の BVH をレイ トレーシング法に適用した．BVH のコストを最小化するように，1 つずつ primitive を挿入していくことで構築する手法である．Leutenegger ら [65] は，ボトムアップの構築法として，ソートを用いた手法を提案している．Muller ら [78] は，ラジオシティ法でのクラスタリングに AABB tree を利用した．Zachmann ら [119] は OBB tree を用いて，リアルタイムの衝突判定アルゴリズムに応用した．VVH が分割スキームとして提案されている．Wachter ら [109] は BIH (Bounding Internal Hierarchy) という手法を提案した．BIH では，BVH の各 BV をデータ構造として持たず，分割平面に相当する 2 つの平面のみをデータ構造として保持する．また，構築の際には global space partitioning heuristic とよばれる戦略を用いており，kd-tree にも近い分割手法といえる．

さらに，リアルタイムレイトレーシングにおいても，BVH の適用例は多く，並列アルゴリズムの開発も進んでいる．レイ トレーシング法で用いられる BVH はそのほとんどが AABB tree である．Wald ら [110] は，kd-tree に適応されてきた分割手法を BVH に適用し，比較することで，kd-tree に対する BVH の優位性を実証した．Wald らの手法では高度な並列化が実現している．Dammertz ら [19] は，QBVH (Quadric Bounding Volume Hierarchy) という手法を提案した．これは，近年の CPU などに搭載されているベクトル演算ユニット用の拡張命令セットである SSE (Streaming SIMD Extension) や AltiVec を利用して，4 つの光線を一度に処理することのできる BVH データ構造である．構築やリーフにおける交差判定にも SIMD (Single Instruction Multiple Data) が利用されている．Tsakok ら [104] は，packet tracing の考

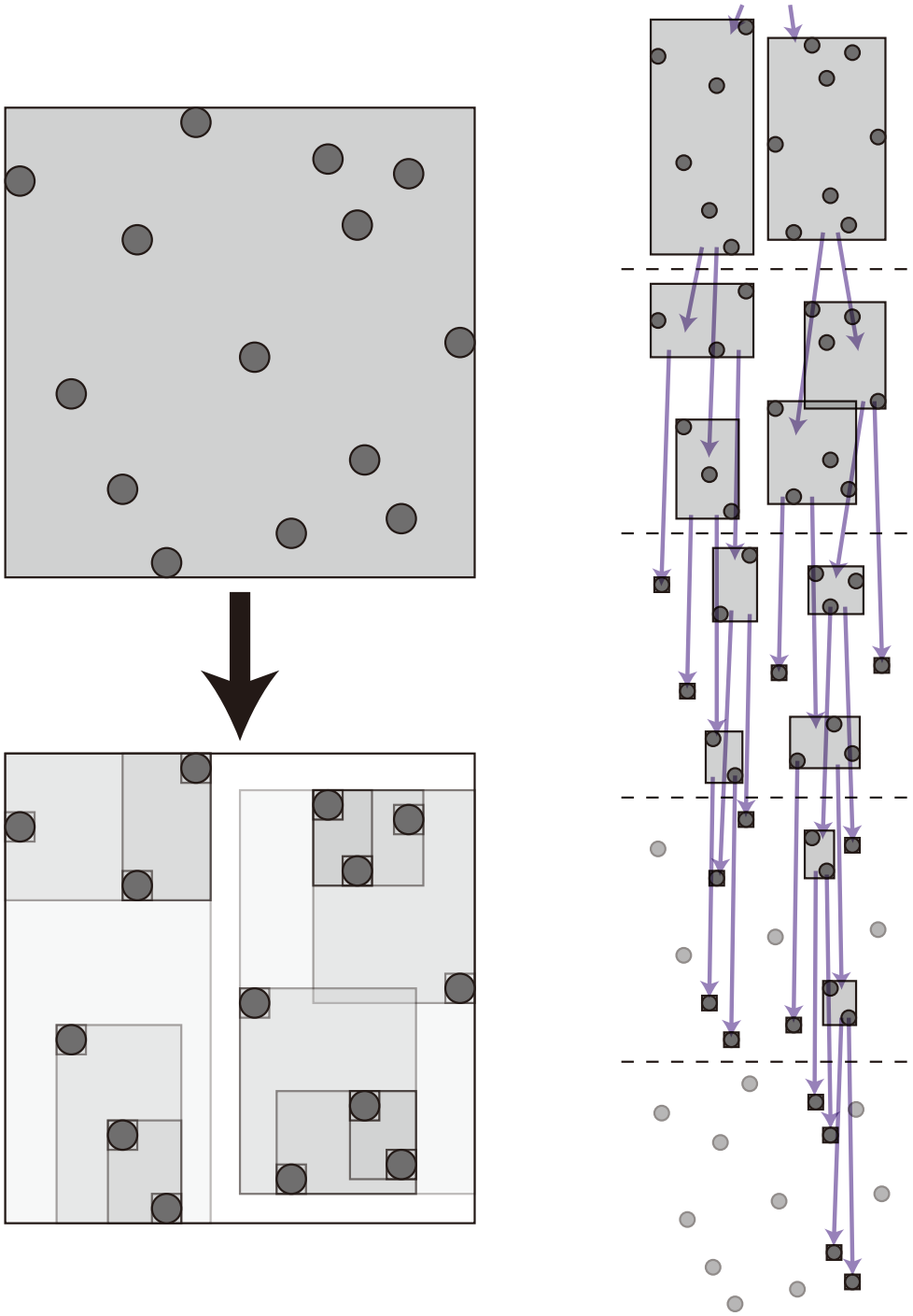


Fig. 2.10　2 次元 BVH のトップダウン構築．ここでは，spatial median を分割スキームとして採用している．

え方を BVH に応用し，SIMD 拡張を行っている．Stich ら [102] は，SBVH (Split Bounding Volume Hierarchy) とよばれる分割手法を提案した．細長い primitive や周囲に比べて極端に大きな primitive があると極端に効率が低下するという BVH の問題点を解消するために，一つの primitive を分割して複数の BV にまたがることを許可して BVH 構築する手法である．構築手法に関して，Lauterbach ら [63] は GPU を用いて高速に AABB tree を構築する手法を提案した．これは GPU-BVH とよばれ，Morton コードを用いて，データ並列に構築する手法である．

2.2　本研究の位置づけ

Wald ら [110] により，高度に core 並列化されたレイ トレーシング法では BVH は kd-tree に比べ効率性において優位に働くことが実証されている．この背景を踏まえ，本研究では空間データ構造に関して BVH を採用する．GPU などのデータ並列環境において，BVH を構築する手法としては Lauterbach ら [63] の GPU-BVH があるが，線形に構築することから，分布が一様でない場合には BVH の質が低下してしまうことが問題としてあげられる．そこで，この線形に構築する手法 (LBVH; Linear Bounding Volume Hierarcy) をその構築手法として参考にしつつ，primitive の空間的局在性を考慮に入れた，新たな BVH の構築手法を提案する．空間的局在性を考慮に入れるため，LBVH の線形性をゆがめることから提案手法を DBVH (Distorted Bounding Volume Hierarchy) とよぶ.

第 3 章

DBVH の提案

本章では，DBVH の提案を行う．

初めに，3.1 節で元となる手法である LBVH について，Lauterbach ら [63] の研究に基づき説明する．その後，3.2 節で DBVH の概要を示したのち，3.3 節でそのアルゴリズムの詳細について説明を行う．

3.1 LBVH

LBVH は AABB を BV として用いる BVH である．LBVH[63] では，BVH をソート問題に落とし込むことで，並列に処理している．

まず，primitive の代表座標をもとに，Morton コード (Z-order, Z-curve) を生成する．代表座標としては, primitive の AABB の中心座標が用いられる．座標を $2^k \times 2^k \times 2^k$ の uniform grid におけるインデックスに換算し，その voxel に対応付けられた Morton コードを各 primitive に割り当てる．Morton コードは (Fig.3.1) に示されるような空間充填曲線 (space filling curve)[*1]であり，各 voxel を $3k$-bit の値で表すことができる．一辺 2^k の uniform grid における，voxel $\boldsymbol{v} = (v_x, v_y, v_z)$ (ただし，$0 \leq v_x \leq 2^k - 1, 0 \leq v_y \leq 2^k - 1, 0 \leq v_z \leq 2^k - 1$) の Morton コード $c(\boldsymbol{v})$ は，以下の式で求められる．

$$c(\boldsymbol{v}) = \sum_{i=0}^{k} \left[v_x^{(i)} 2^{3i+2} + v_y^{(i)} 2^{3i+1} + v_z^{(i)} 2^{3i} \right] \tag{3.1}$$

ただし， $x^{(i)}$ は， x の二進数表記における下から i 桁目の値を表すものとする．

[*1] そのほかの空間充填曲線として Hilbert 曲線やドラゴン曲線，Sierpiński 曲線などが有名である．

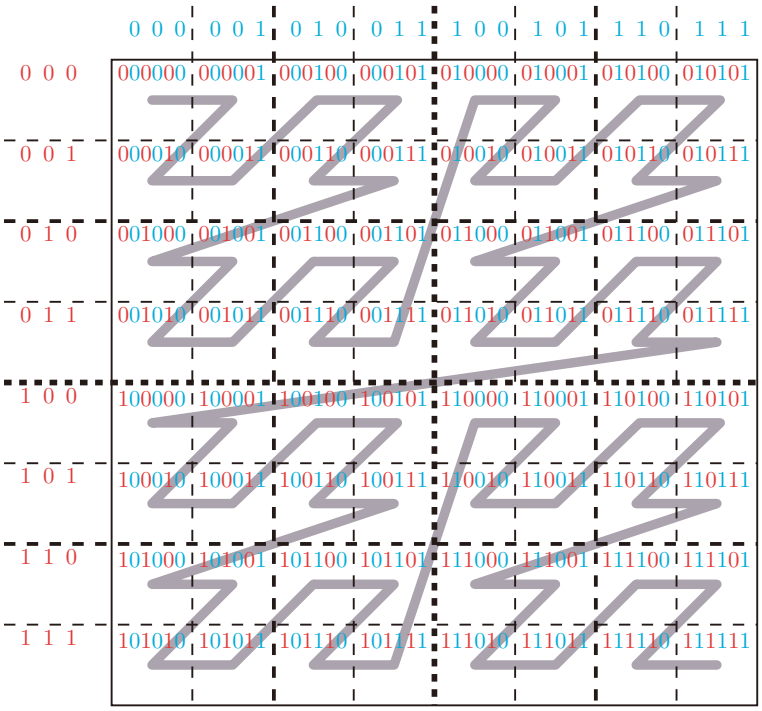


Fig. 3.1　2 次元の Morton コード．x 3bit, y 3bit の 6-bit Morton コードである．その順につないだ線分の集合に Z が大量に現れることから Z curve ともよばれる．

その後，primitive を Morton コードをキーとしてソートする．Lauterbach らは，GPU でのソートの実装として，GPU radix sort[97] を採用している．ソート後，隣り合う primitive i と $i+1$ について，Morton コードを上位の bit から比較して，最初に異なる bit が何番目であるかを計算する．ここでその値を split level h_i とよぶ．split level h_i を用いて，split list $[(i,h_i),(i.h_i+1),\ldots,(i,3k)]$ を生成する．生成した split list をすべての primitive について結合し，$[(0,h_0),\ldots,(0,3k),(1,h_1),\ldots,(1,3k),\ldots(n,3k)]$ を得る．これを，level をキーとして再びソートをおこなうことで，その level における BVH ノードが把握できる．(Fig.3.2) にそのアルゴリズムの適応例を示す．

LBVH によって生成された木は 2 分木ではなく，一つしか子の存在しないノード (singleton node) が大量に存在する木が生成される．そこで，階層構造を構築したのち，LBVH ではこれを取り除く後処理が施される．

最終的に，木構造をリーフ ノードから AABB の結合を求めながらボトムアップに走査していくことで，各ノードにおける AABB の値を更新する．

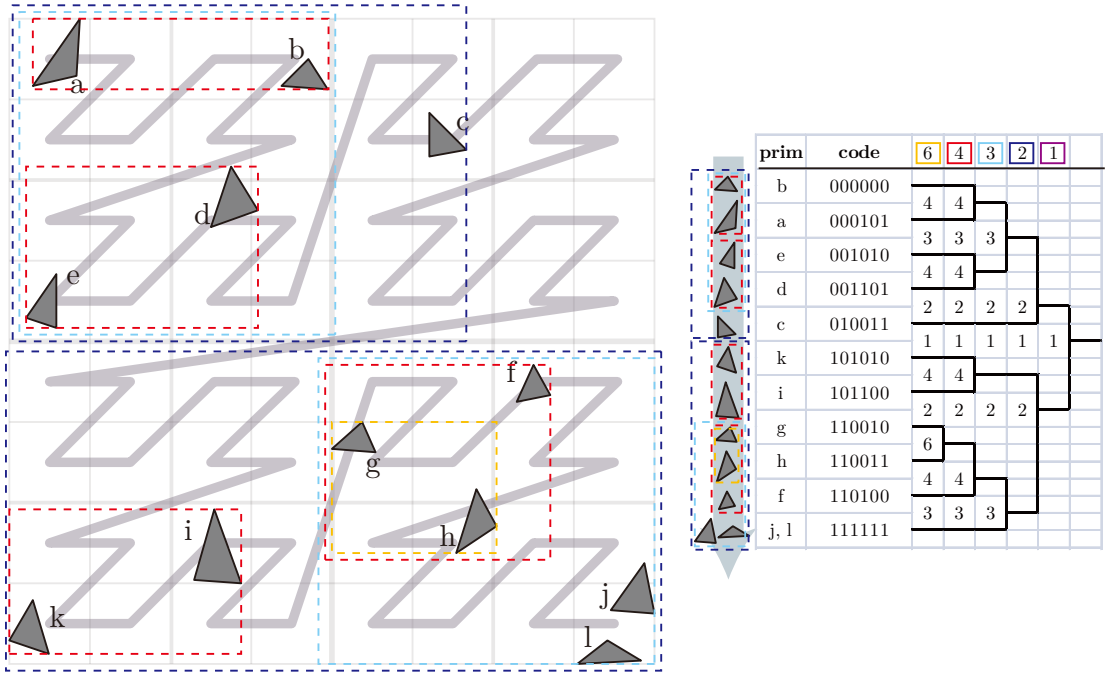


Fig. 3.2　LBVH の構築手法．a〜l の各 primitive について，Morton コードを求め，右表のようにソートする．その後，split list を作ることで，その level でのノードがわかる．右表では level 5 での split にあたる primitive が存在しなかったため，省略してある．

3.2　DBVH の概要

LBVH による構築手法では，初めの Morton コードの生成の際に一様な空間分割手法に基づいて空間的コヒーレンスを実現している．そのため，空間的局在性の大きいシーンでは，BVH の品質が低下してしまうという問題点がある．Lauterbach ら [63] によると，レイ トレーシング法における効率は SAH で構築した BVH にくらべ 10 % 〜40 % [*2] の効率にとどまっている．そこで，初めに LBVH で荒く空間分割を行ったのち，それぞれのリーフノードについて SAH を用いてトップダウンに構築する hybrid な手法を導入，提案している．SAH を用いてトップダウンに構築する手法は，並列型のアーキテクチャと親和性が低く，LBVH に比べて GPU において 10〜40 倍の実行時間がかかる．そのため，hybrid な手法では，LBVH の数倍の構築時間を費やすことになる．

[*2] ただし，Bunny および Dragon については 94 % の効率となっている．

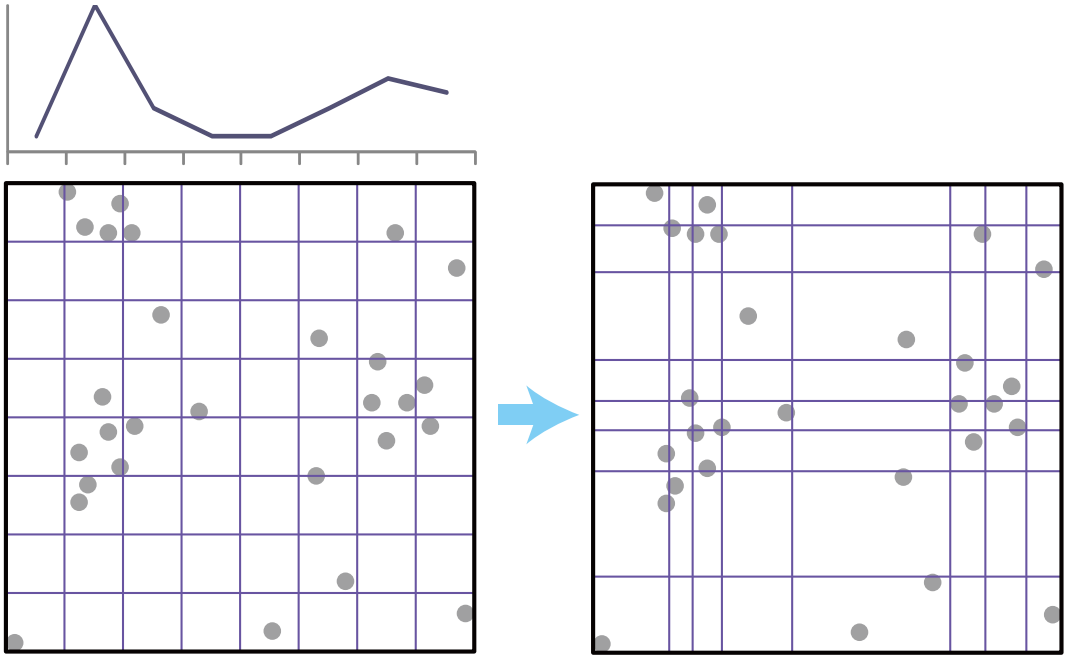


Fig. 3.3　DBVH の概念図．LBVH(左図) では一様に grid を生成しているが，primitive の分布によってヒストグラムを作成し，DBVH(右図) ではゆがんだ grid を生成する．

そこで，DBVH では，ボトムアップでの構築手法における BVH の品質を向上させることに主眼をおく．DBVH での BVH の品質は，LBVH での Morton コードを求める前に，ヒストグラムを用いて，空間を不均一に voxel 分割し，Morton コードをゆがめることで空間的局在性を考慮に入れた BVH の構築を行う．ここで，このヒストグラムを用いた不均一格子の生成方を，ヒストグラム歪曲化 (histogram distortion) とよぶ．この，概念図を (Fig.3.3) に掲載する．

3.3　DBVH のアルゴリズム

まず，primitive の代表座標を用いてヒストグラムを生成する．最終的に生成する Morton コードは $3k$-bit であるため，$2^k \times 2^k \times 2^k$ よりも細かい解像度でヒストグラムを生成することが望ましい．この解像度は任意に取ることができる．そこで，任意の自然数 α を用いて，ヒストグラムの一辺の解像度を $2^k\alpha$ と表す．まず，0 で初期化された長さ $2^k\alpha$ の配列を各軸について確保する．各 primitive について，代表座標から voxel を計算し，対応する配列の値に 1 を加える．これを voting または bucketting とよぶ．その後，この配列に対して prefix

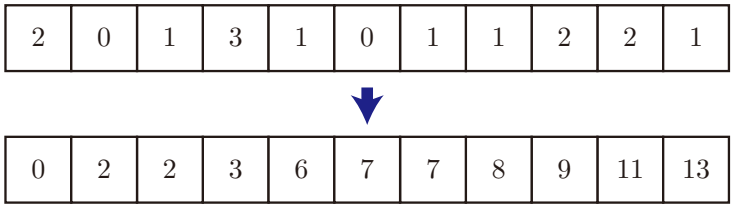


Fig. 3.4　prefix scan の例．逐次的な処理の場合，prefix scan では次々と値を足し合わせていくことで簡単に実現できる．

scan (prefix sum) を行うことで，ヒストグラムを生成することができる．prefix scan は数列 $\{a_0, a_1, a_2, \ldots, a_i, \ldots, a_n\}$ に対して，

$$\left\{0,\ a_0,\ a_0 + a_1,\ \ldots,\ \sum_{k=0}^{i-1} a_k,\ \ldots,\ \sum_{k=0}^{n-1} a_k\right\} \tag{3.2}$$

を求める操作である．prefix scan の例を，(Fig.3.4) に示す．この prefix scan については，GPU 等における並列アルゴリズムが考案されており，parallel prefix scan[43] とよばれる．

次に，生成したヒストグラムに基づいて，Morton コードをゆがめる．各 primitive i について自分が vote した voxel のヒストグラムの値を参照する．このヒストグラム値を g_i とすると，$0 \leq g_i \leq n-1$ を満たす．ここで n は全 primitive の数である．求める $3k$-bit の Morton コードは，g_i を用いて，

$$c(\lfloor 2^k \frac{g_i}{n} \rfloor) \tag{3.3}$$

とあらわされる．

その後の計算の流れは，LBVH と同様である．計算した Morton コードをキーとして primitive をソートし，隣接 primitive について split list を構築する．すべての primitive について split list を結合したのち，level をキーとして再ソートし，得られる split list を参照し，木構造を構築する．

LBVH 同様，singleton node が生成されるため，これを取り除く処理を行い，AABB を更新することで BVH の構築が完成する．

第 4 章

DBVH の実装

4.1 CPU における DBVH の実装

前章で述べた DBVH の構築手法をまとめると (Fig.4.1) のようになる．
ここでは，プログラミング言語として C++ を用い，CPU による実装を説明する．

ヒストグラム用の配列を用意

ヒストグラム用の配列は $2^k\alpha + 1$ の大きさのものを x, y, z それぞれに用意し，0 で初期化しておく．1 つ余計に持つのは exclusive prefix scan[*1] を実現するために，値が 0 の余分なデータを先頭に付与するからである．

ヒストグラムへの voting

ヒストグラムへの voting は各 primitive についての処理である．各 primitive について for ループを回し，代表座標に対する voxel の位置を求め，ヒストグラム用の配列に voting する．再度ヒストグラムにアクセスすることを想定し，このときの voxel の位置は再計算を防ぐために，別の配列に格納しておくとよい．また，exclusive prefix sum を実現するため voting の際の voxel の位置も 1 つずれることに注意されたい．

prefix scan

CPU における prefix scan は，配列の頭から順にデータを走査し，i 番目の値を $i + 1$ 番目の値に加える処理を行うことで実現する．

[*1] exclusive prefix scan とは $0, a_0, a_0 + a_1, \ldots, \sum_{k=0}^{n-1} a_k$ と先頭が 0 で始まる prefix scan である．このとき，最後のデータは使用されない．対象的に inclusive prefix scan では $a_0, a_0 + a_1, \ldots, \sum_{k=0}^{n} a_k$ となる．

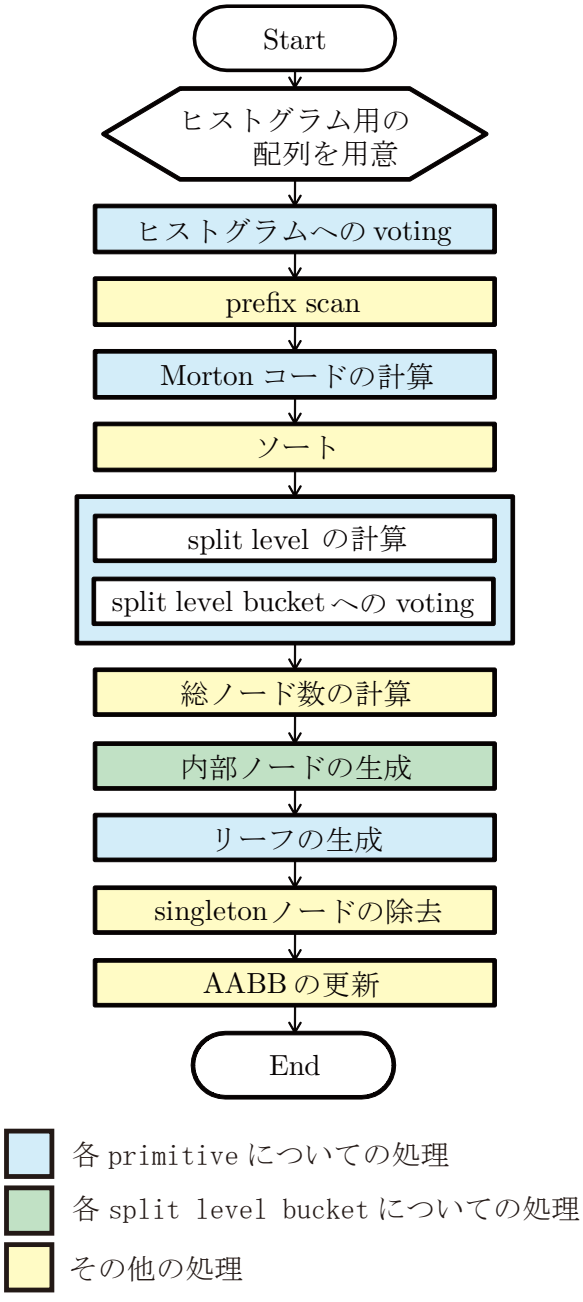


Fig. 4.1 DBVH 構築のフローチャート.

Morton コードの計算

ここまでで計算したヒストグラムを各 primitive から参照することで，ゆがませた格子での Morton コードを計算する．Morton コードの計算については，実際のソースコードを (List 4.1) に示す.

ソート

ソートについてはここでは STL (Standard Template Library) の sort 関数を用いる．

split level の計算

split level の計算は，隣接する 2 つの primitive の Morton コードについて，上位ビットから走査し，初めて異なる bit を示した bit を探す操作である．ここでは，XOR のビット演算を使って実装を行う．実際のソースコードを (List 4.2) に示す.

List 4.1　Morton コードの計算

```
unsigned int ComputeMortonCode(unsigned int x, unsigned int y, unsigned int z)
{
  unsigned int u = 0;
  for (unsigned char i = 0; i < K; ++i)
  {
    if (x & (1 << i)) u |= (1 << (3*i+2));
    if (y & (1 << i)) u |= (1 << (3*i+1));
    if (z & (1 << i)) u |= (1 << (3*i+0));
  }
  return u;
}
```

List 4.2　split level の計算

```
unsigned int MostSignificantBitTrueInXor(unsigned int a, unsigned int b)
{
  unsigned int x = a ^ b;
  for (unsigned int i = K * 3; i > 0; --i)
  {
    if (!x)
      return i;
    x >>= 1;
  }
  return 0;
}
```

split list bucket への voting

split list を結合してソートする代わりに，ここでは $3k$ 個の bucket (split level bucket) を用いて，各 primitive から該当する level の bucket に primitive へのインデックスを格納する．

総ノード数の計算

総ノード数は，$3k$ 個の bucket のサイズの総和として算出する．算出した総ノード数分のバッファを木構造構築のために確保する．

内部ノードの生成

level i と level $i+1$ の bucket を参照することで，そのノードが子を 1 つもつか，2 つもつかを判断することができる．この情報をもとに level の低いほう (木のルートに近いほう) から順に内部ノードを生成する．

リーフの生成

最終的に各 primitive についてリーフを生成する．AABB が一致していたり，primitive の代表点が近すぎたりして，$3k$ bit の Morton コードで分けることができなかった primitive が存在する場合は，1 つのリーフに複数の primitive が格納される．実装の方法によっては，ここで新たに内部ノードを生成し，1 リーフ 1primitive を達成するまでリーフを分割する方法を採用してもよい．

singleton ノードの除去

リーフ以外のすべてのノードが 2 つの子をもつように，木構造の最適化を行う．1 つしか子がないノードは，その子ノードのデータをコピーし，その子ノードに不使用のフラグを付加する．この際，不使用なノードの数を数えておき，先ほど計算した総ノード数から差し引くことで，最終的な木構造でのノード数を算出する．traversal に用いる最終的な木構造では，メモリアクセスにおけるキャッシュのヒット率を上げるため，メモリのアラインメントを考慮してメモリの割り当てを行う．リーフ以外のすべてのノードが 2 つの子をもち，その 2 つの子ノードがデータ上に連続して並んでいることを前提とすることで，子ノードへのポインタは 1 つだけの保持で済む．データ構造の一例を (Table4.1) に示す．その後，不使用のフラグが付加されていないノードのみ子ノードのインデックスまたは primitive へのインデックス情報をコピーする．

Table 4.1　DBVH のノードのデータ構造の例．union を用いることで，32 byte に収めることができる．ただし，32 bit 環境のとき．

変数		用途	サイズ
float3 aabbMin		AABB の最小値	12 byte
float3 aabbMax		AABB の最大値	12 byte
union	unsigned int flag	内部ノードかリーフかの判別	4 byte
	unsigned int nPrimitive	リーフ内の primitive の数	
union	Node* child	子ノードへのポインタ	4 byte
	Primitive* onePrim	リーフ内の primitive へのポインタ	
	Primitive** multiPrims	リーフ内の primitive の配列へのポインタ	

AABB の更新

AABB の結合を計算しながら，再帰的に AABB の更新を行うことができる．

4.2　ヒストグラム歪曲化の派生的手法の実装

BVH の品質が極度に低下する要因の一つに，そのシーンにおいて，primitive の大きさにばらつきがある場合があげられる．実際にモデリングされたシーンでは，壁などの大きな平面は 1 枚のプレーン (plane) としてモデリングすることが多く，他の primitive に比べ大きな primitive となりやすい．これらの状況で，BVH の構築を行うと，大きな AABB に周りの primitive が吸収されてしまい，品質の低下につながってしまう．そこで，本節では primitive の大きさをヒストグラム生成の際に考慮にいれることによって，このような状況を緩和する手法に関して議論を行う．

DBVH は，ヒストグラムの生成により格子をゆがめることにより，空間的局所性を考慮に入れる点が特徴である．ここまで，ヒストグラム歪曲化として，primitive の代表点を用い，その primitive の代表点が存在する配列要素に 1 を加えてヒストグラムを構成するという手法を提案した．しかし，primitive の代表点を対象としたヒストグラムの構成法以外にも，次のような派生的手法が考えられる．

AABB の各辺の長さを利用した手法 (axis delta 法)　primitive を用いてヒストグラムを構成する際に，各 primitve の AABB における各辺の長さを利用する手法である．prim-

itive i の AABB[*2]における最小点を $\boldsymbol{vmin}^i = (xmin, ymin, zmin)$，最大点を $\boldsymbol{vmax}^i = (xmax, ymax, zmax)$ とすると，各辺の長さ $\boldsymbol{d}^i$ は次式によって得られる．

$$\boldsymbol{d}^i = \boldsymbol{vmax}^i - \boldsymbol{vmin}^i \tag{4.1}$$

この値を指標として用い，代表点が存在する配列要素に $\boldsymbol{d}^i$ を加えることでヒストグラムを構成する．ヒストグラムは x 軸，y 軸，z 軸の各軸について生成するため，$\boldsymbol{d}^i$ の各成分を利用する．求める Morton コードは $S = \sum_{i=0}^{n} d^i$ 及び，ヒストグラム値 g_i を用いて，

$$c(\lfloor 2^k \frac{g_i}{S} \rfloor) \tag{4.2}$$

とあらわされる．

AABB の各面の大きさを利用した手法 (orthogonal area 法)　primitive を用いてヒストグラムを構成する際に，各 primitve の面の大きさを利用した手法である．primitive の AABB における x 軸，y 軸，z 軸に直交する面の大きさ $\boldsymbol{A}^i$ は次式によって得られる．

$$\boldsymbol{A}_i = (d_y^i d_z^i, d_z^i d_x^i, d_x^i d_y^i) \tag{4.3}$$

この値を指標として用い，代表点が存在する x 軸，y 軸，z 軸の各配列要素に A_x^i，A_y^i，A_z^i を加えることによりヒストグラムを生成する．axis delta 法同様に，ヒストグラムを参照し Morton コードを参照し，格子をゆがめる．

AABB の表面積を利用した手法 (surface area 法)　primitive を用いてヒストグラムを構成する際に，各 primitve の AABB における表面積を利用する方法である．表面積は，$\boldsymbol{d}^i$ を用いて，

$$2(d_x^i d_y^i + d_y^i d_z^i + d_z^i d_x^i) \tag{4.4}$$

とあらわされる．最終的には比率を用いるので，係数の 2 は省略し，指標として，

$$d_x^i d_y^i + d_y^i d_z^i + d_z^i d_x^i \tag{4.5}$$

を用いる．前述の 2 つの手法同様に，ヒストグラムを参照し Morton コードを参照し，格子をゆがめる．

[*2] AABB については，次章 5.1.3 項においても詳しく述べる

AABB の体積を利用した手法 (volume 法)　primitive を用いてヒストグラムを構成する際に，各 primitve の AABB における体積 V^i を利用する手法である．体積 V^i は以下の式を用いて算出する．

$$V_i = d_x^i d_y^i d_z^i \tag{4.6}$$

この値を指標として用い，代表点が存在する配列要素に $\boldsymbol{d}_i$ を加えることでヒストグラムを構成する．

第 5 章

DBVH のリアルタイムアプリケーションに向けた応用

5.1 レイ トレーシング法への応用

先に述べたように，レイ トレーシング法においては，その処理の大部分が，光線とシーンとの交差判定 (ray-scene intersection test) に費やされる．BVH の階層構造を利用して，枝刈りをすることで，この交差判定を効率化することができる．

5.1.1 光線の表現と ray-scene intersection test

一般に，光線 R はパラメータ $t \in \mathbb{R}$ を用いて以下のようにあらわされる．

$$R = \left\{ \boldsymbol{r}(t) \mid \boldsymbol{r}(t) = \boldsymbol{o} + t\boldsymbol{d},\ t_{\min} \leq t \leq t_{\max} \right\} \tag{5.1}$$

ここで，$\boldsymbol{o} \in \mathbb{R}^3$ は 3 次元空間上の点，$\boldsymbol{d} \in \mathbb{R}^3$ は方向ベクトル[*1]を表す．

レイ トレーシング法で必要となる ray-scene intersection test には一般に 2 種類ある．光線 R とシーン S の交点 $R \cap S = \{\boldsymbol{r}(t_0), \boldsymbol{r}(t_1), \boldsymbol{r}(t_2), \ldots\}$ のうち，最小の t_i についての $\boldsymbol{r}(t_i)$ を求める問題と，光線 R がシーン S と交わるかどうかさえ判断すればよい (すなわち $R \cap S = \varnothing$ or not) 問題の 2 種類である．前者は，(Algorithm 1) の「光線 *ray* に対してシーンと一番近い交点を求める」処理に用いられ，後者は「影光線 sray がシーンと交わ」るかどうかの判断 (occlusion test) に用いられる．本研究では，この 2 種類に対して別の query 関数を用意して対応する．

[*1] 一般に正規化されていることが多い．

Algorithm 1 Whitted レイ トレーシング

シーンを読み込む.
acceleration structure を構築する.
for all ピクセル サンプル *i* **do**
　光線 *r* を生成する.
　L_i += TraceRay (*r*)
end for 後処理など.

Procedure: TraceRay (*ray*)
光線 *ray* に対してシーンと一番近い交点を求める.
影光線 *sray* を生成する.
if 影光線 *sray* がシーンと交わらない **then**
　光源からの寄与を考慮した表面の色 *l* を計算.
end if
while *rayDepth* + + ≤ (最大光線深度) **do**
　if 表面で反射・屈折がおこる **then**
　　光線 *ray* を生成する.
　　l += TraceRay (*ray*)
　end if
end while
return *l*

5.1.2　BVH の traversal

accerelation structure として，BVH を採用した場合の traversal について，現在 stack を用いた手法と stackless の手法が存在する．(Fig.5.1) に stack を用いた場合のフロー チャートを掲載する．ここでは，厳密な交点を求める必要がある場合のフロー チャートとなっており，primitive と hit した場合には光線の $t_{\max}$ を hit した場所の *t* に置き換え，さらにループを続ける．一方，影光線に対する occlusion test では，シーンと交差するかどうかだけに興味があるので，初めて primitive の交差判定が hit した時点で true を返し，この処理を終了すればよい.

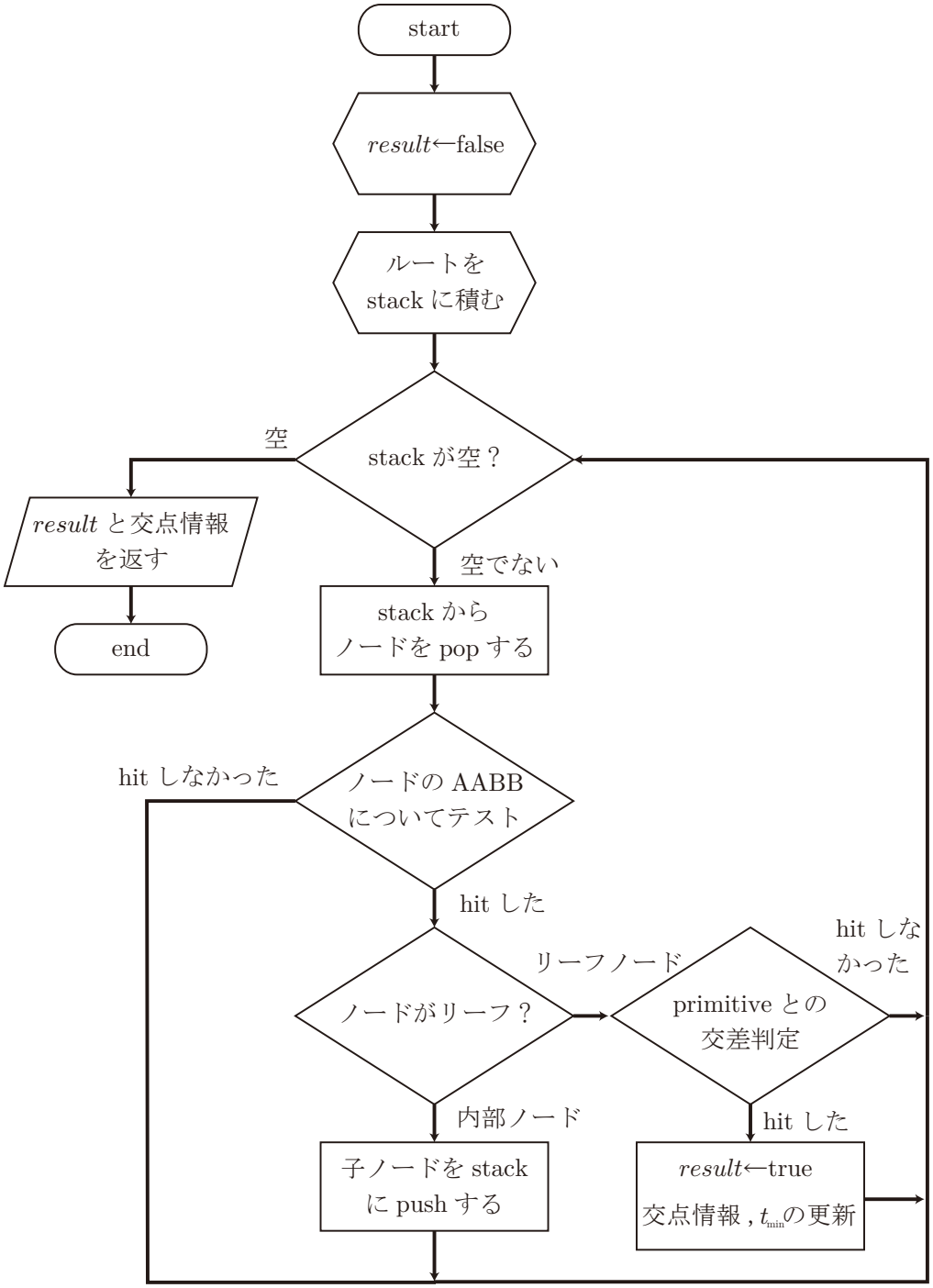


Fig. 5.1　BVH traversal のフロー チャート.

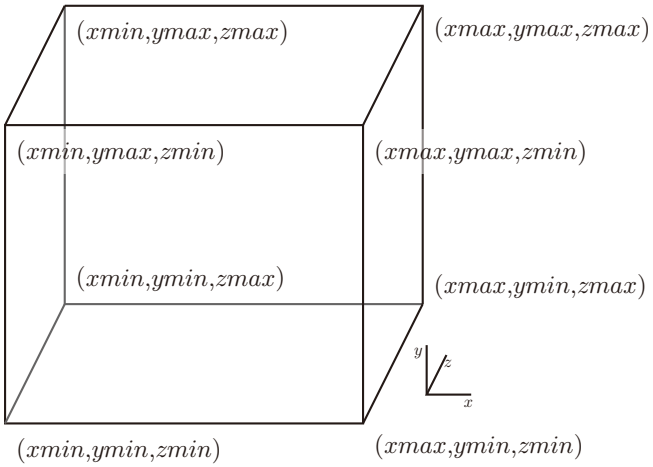


Fig. 5.2　AABB.

5.1.3　ray-AABB テスト

BVH のノードについてのテストは，その BV と光線の交差判定を行うことになる．DBVH は AABB を BV とする空間分割手法であるので，ここで用いられる交差判定は，光線と AABB の交差判定 (ray-AABB test, ray-box test) になる．AABB は，(Fig.5.2) に示すように，$xmin, ymin, zmin, xmax, ymax, zmax$ の 6 つの浮動小数であらわすことができる．

よく用いられる ray-AABB test の手法に，slab 法 (slab method, slab test) [56] がある．これは，AABB を構成する x，y，z の各軸に垂直な 3 ペアの平面 (slab) によって，光線が切り取られる各軸に沿った距離 3 つを求めることによって判定を行う手法である．まず，slab test の最初のテストは，光線と各平面との交点を求めることである．例えば，$xmin$ を通り，x 軸に垂直な平面と光線との交点 $\boldsymbol{r}(t_{xmin})$ は以下のように求められる．

$$\boldsymbol{r}(t_{xmin}) = \boldsymbol{o} + \boldsymbol{d}t_{xmin} \tag{5.2}$$

$$\therefore r(t_{xmin})_x = o_x + d_x t_{xmin} = xmin \tag{5.3}$$

$$t_{xmin} = \frac{xmin - o_x}{d_x} \tag{5.4}$$

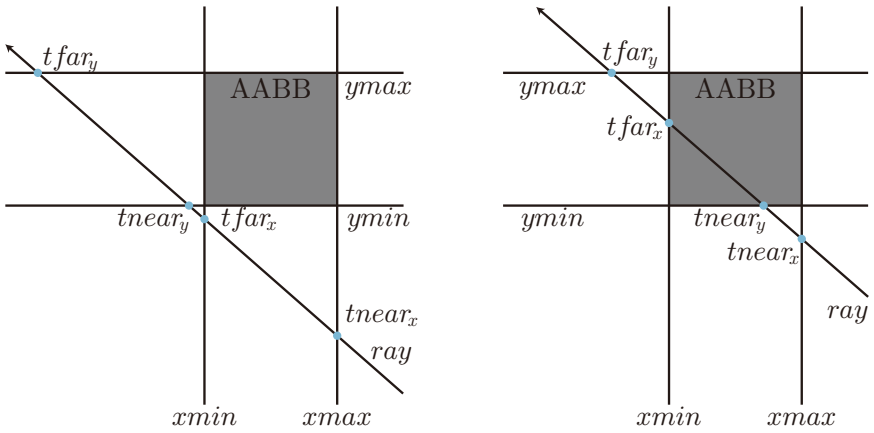


Fig. 5.3　2 次元での slab test の例．左図では，$tnear \leq tfar$ が満たされていない．図でも AABB と *ray* が交差していないことが分かる．一方，右図では式 (5.8) の 3 式を満たしている．図でも AABB と *ray* が交差していることが確認できる．

ここで，割り算を避けるために[*2]，BVH の traversal 前に光線の方向ベクトルについて各成分の逆数 d_x^{-1}，d_y^{-1}，d_z^{-1} を求めておき，これを乗算することで対応する．一般にこれを，IRDVC (Inverse Ray Direction Vector Components) とよぶ．

$$t_{xmin} = (xmin - o_x)d_x^{-1} \tag{5.5}$$

同様にして，$t_{xmax}, t_{ymin}, t_{ymax}, t_{zmin}, t_{zmax}$ についても求める．次に，t_{xmin} と t_{xmax} のうち，値の小さいものを $tnear_x$，大きいものを $tfar_x$ に格納する．$tnear_y$，$tfar_y$，$tnear_z$，$tfar_z$ も同様の手順で求め，最終的に以下の式で *tnear* 及び $tfar$ を得る．

$$tnear = \max(tnear_x, tnear_y, tnear_z) \tag{5.6}$$
$$tfar = \min(tfar_x, tfar_y, tfar_z) \tag{5.7}$$

これらの値に対して，次の 3 式を満たせば，その光線 R は AABB と交差していると判断できる．

[*2] 一般に割り算の命令は遅いため．

$$\begin{cases} t_{\min} \leq tfar \\ tnear \leq tfar \\ tnear \leq t_{\max} \end{cases} \tag{5.8}$$

2 次元での slab test の例を (Fig.5.3) に示す.

第 6 章

結果と考察

6.1 空間分割手法の比較実験

本研究では，以下の空間分割手法を実装し，比較実験を行った．

6.1.1 空間分割手法の実験設定

MBVH トップダウン構築による BVH の比較対象として，space median 戦略により分割する BVH を選択した．以下，これを MBVH と表記する．中点で 2 つに分割した際に，どちらか片方に primitive が一つも含まれなかった場合は，含まれる primitive をソートし，object median を分割戦略として採用する．traversal には，SSE を用いた stack ベースの手法を用いる．

LBVH LBVH では，構築に用いられる Morton コードの bit 数 k がパラメータとなる．以下では，x，y，z 成分それぞれに同じ bit 数 k を採用した，3k-bit の Morton コードを用いて構築した LBVH を LBVH[k] と表記する．結果として提示するほとんどの例で 10 bit の Morton コードを使用している．LBVH の構築段階で，primitive が近すぎる及び AABB が等しいために Morton コードによって区別できなかった primitive については，1 リーフに複数の primitive が存在することを許容する．traversal には，SSE を用いた stack ベースの手法を用いる．

DBVH MBVH では，構築に用いられる Morton コードの bit 数 k およびヒストグラム生成の際の細分化係数 α (3.3 節参照) がパラメータとなる．以下では，α について，$\alpha = 2^s$ となるようにとり，s，k のパラメータを用いて構築した DBVH を DBVH[k, s] と表記する．結果として提示するほとんどの例で 10 bit の Morton コードを使用している．

Table 6.1　実験に用いたモデルデータ．すべてのポリゴンは三角形で構成されるようにモデルデータを変更した．

名称	ポリゴン数	備考
chess	3,592	
knot	11,616	
happy buddha	100,000	以下 buddha と表記.
渦貝	180,533	以下 uzu と表記.
maxplanck	50,801	
Spontza Atrium	279,163	以下 spontza と表記.

DBVH の構築段階で，primitive が近すぎる及び AABB が等しいために Morton コードによって区別できなかった primitive については，1 リーフに複数の primitive が存在することを許容する．traversal には，SSE を用いた stack ベースの手法を用いる．

6.1.2　実験モデル データ

実験に用いたモデルデータの名称及びポリゴン数を，(Table6.1) にまとめる．モデルデータの著作権は作成者に帰属する．なお以後，備考欄に記載した名称でモデルデータを呼称する．また，参考画像を (Fig.6.1) に示す．maxplanck および spontza に関しては，(Fig.6.15) を参考にされたい．

6.1.3　実験環境

以下の実験においては，CPU Intel Core i7 920 (2.67 GHz)，メモリ DDR3 4GB，OS Windows 7 32-bit の計算機を用いた．

6.2　構築速度による比較と考察

本節では，構築速度による比較を行う．

chess，knot，buddha，uzu の各モデルについて，空間分割構造 MBVH，LBVH[10], DBVH[10,2], DBVH[10,3], DBVH[10,4] の構築を行い，その構築に要した時間を計測した．空間構造とモデルの各組について，それぞれ 5 回ずつ計測し，最も遅かったものと最も早

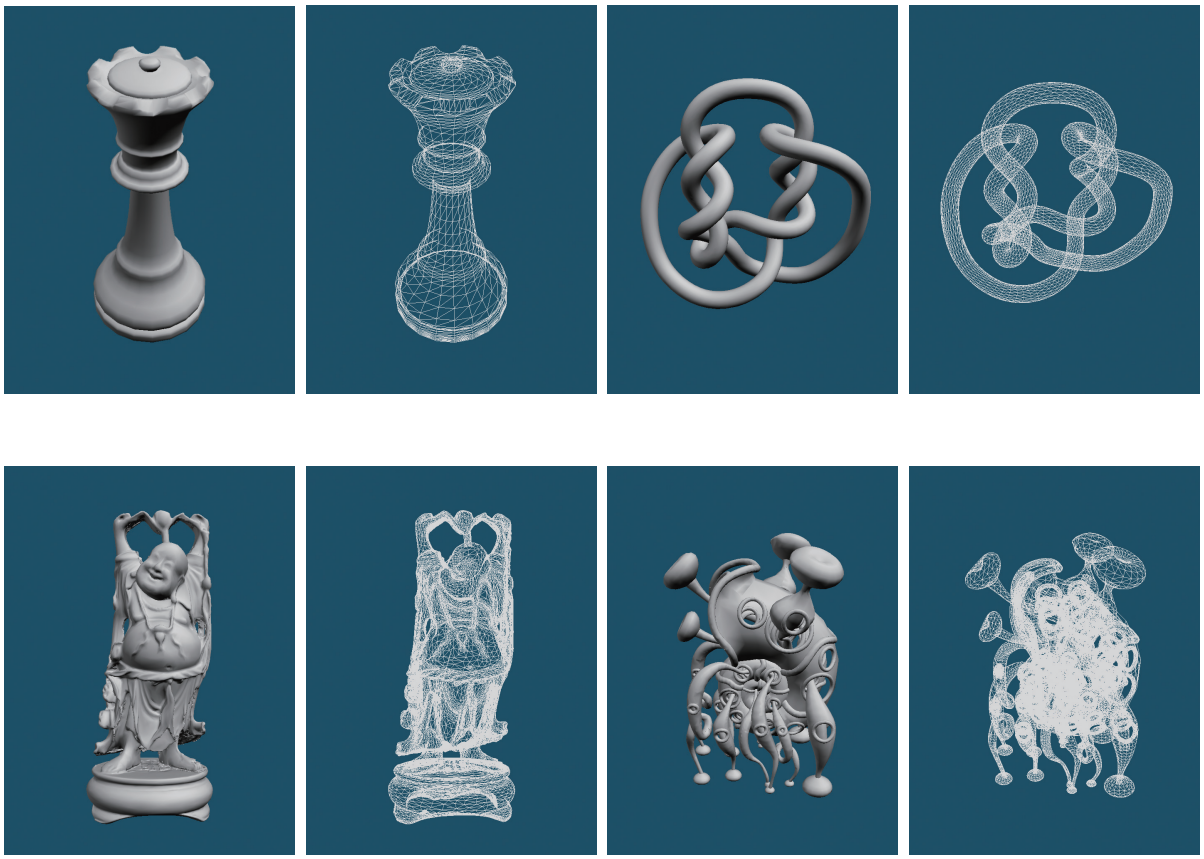


Fig. 6.1　左上から順に，a) chess (ソリッド モデル)，b) chess (ワイヤー フレーム)，c) knot (ソリッド モデル)，d) knot (ワイヤー フレーム)，e) buddha (ソリッド モデル)，f) buddha (ワイヤー フレーム)，g) uzu (ソリッド モデル)，h) uzu (ワイヤー フレーム)

かったものを除いた 3 回の計測の平均値を記録とする．その結果をグラフにまとめたものを (Fig.6.2)，(Fig.6.3)，(Fig.6.4)，(Fig.6.5) に示す．縦軸は構築に要した時間であり，それぞれスケールが異なる点に注意されたい．構築はいずれも CPU での実装である．

DBVH の構築に際し，LBVH に比べて knot モデルでは 6.4 %，chess モデルおよび buddha モデルでは 10 %，uzu モデルでは 15 % 程のオーバーヘッドが存在する．一方，構築の際に用いるヒストグラムの解像度に関しては，その大小でほとんど差がないといえる．CPU での実装ではあるが，トップダウンでの構築に比べて LBVH や DBVH の構築は高速であり，その傾向は primitive 数が多くなるほど顕著である．特に uzu モデルでは 2 倍近い差が出ていることが分かる．

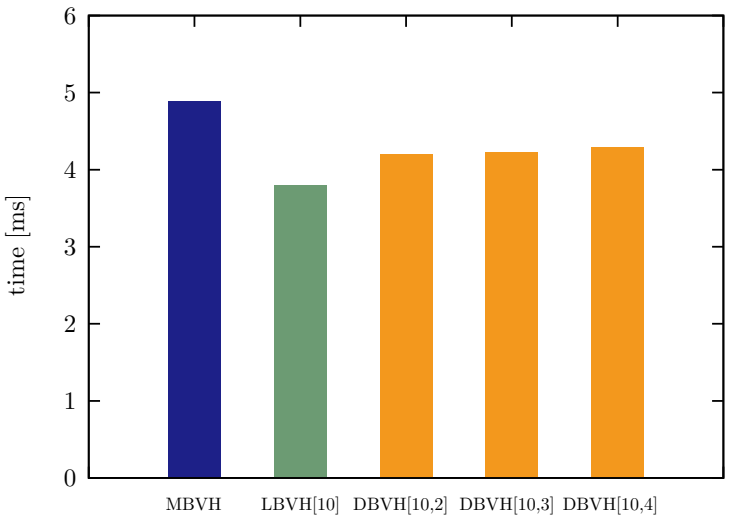


Fig. 6.2　chess モデルにおける空間分割構造の構築時間の比較.

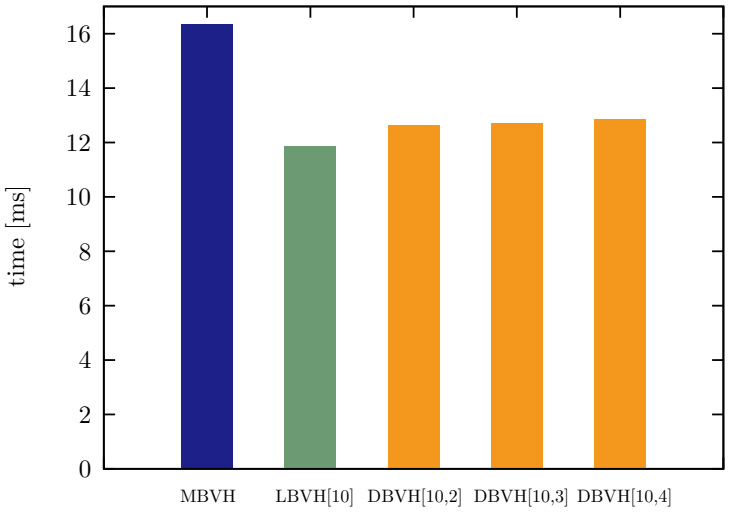


Fig. 6.3　knot モデルにおける空間分割構造の構築時間の比較.

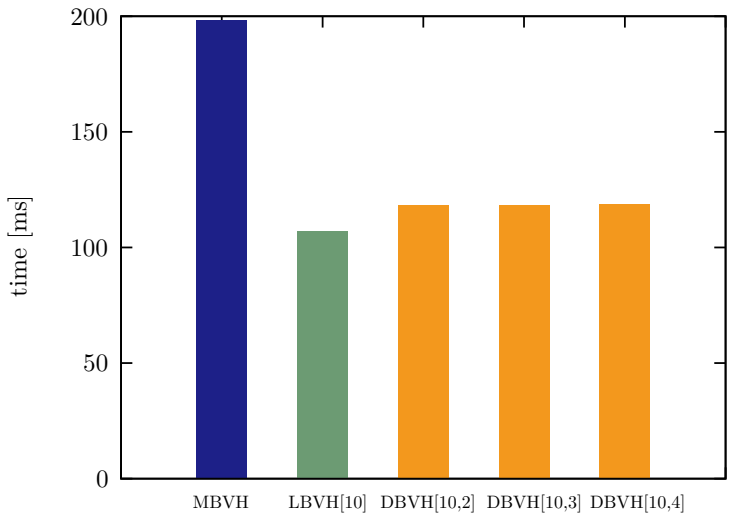


Fig. 6.4　buddha モデルにおける空間分割構造の構築時間の比較.

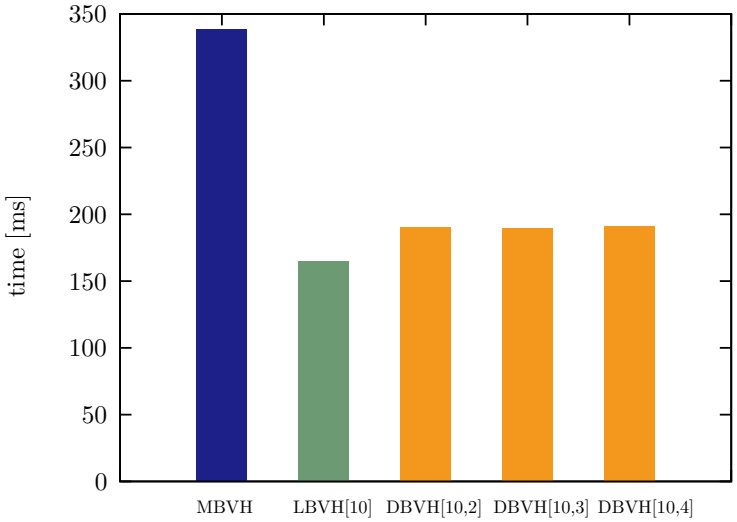


Fig. 6.5　uzu モデルにおける空間分割構造の構築時間の比較.

Table 6.2　1 リーフに分けきれなかった primitive 数の比較.

モデル	LBVH[10]		DBVH[10,2]		DBVH[10,3]		DBVH[10,4]	
chess	76	2.12 %	12	0.33 %	7	0.19 %	5	0.14 %
knot	345	2.97 %	361	3.11 %	357	3.07 %	351	3.02 %
buddha	3,730	3.73 %	2,624	2.62 %	2,538	2.54 %	2,489	2.49 %
uzu	16,938	9.38 %	13,641	7.56 %	13,523	7.49 %	13,471	7.46 %

6.3　品質による比較と考察

まず，MBVH, LBVH, DBVH での構築を行った結果の画像を示す．(Fig.6.6)，(Fig.6.7), (Fig.6.8) は chess モデルを用いて，MBVH，LBVH[10]，DBVH[10,2] を構築した結果の図である．

リーフに到達したノードについてはそれ以上深い階層では描画されていない．MBVH の d 図における駒の胴部や LBVH や DBVH[10,2] の d 図における底部にノードがないのはそのためである．MBVH では深さ 12 以前で，リーフ ノードに達する場合も多く，質の良い BVH が構成されていることが伺える．LBVH と DBVH の比較では，DBVH の方が深さ 3 程度でくびれ部分などの特徴をよくとらえており，深さ 6 ではルークの側面についた溝の形状が判別できる．

LBVH や DBVH では 1 つのリーフ ノードに複数の primitive が格納されていることがある．これは，primitive が非常に密集している場合やポリゴン メッシュの品質が悪い時におきやすい．LBVH と DBVH で 1 リーフに分けきれなかった primitive の数を計算し，比較する．リーフ数が M の二分木では，総ノード数は $2M - 1$ となる．そこで，primitive 数を N とすると，1 リーフ内に 1 primitive しか存在しないように分割した場合，M と N の関係は $M = 2N - 1$ となる．このことから，primitive 数とリーフ数の差，すなわち

$$K = N - \frac{M + 1}{2}$$

で木構造の品質を評価することができる．

(Fig.6.2) に K の値および，その N に対する比率 K/N を百分率で表示した．なお，MBVH においては，すべてのモデルにおいて $K = 0$ になることを確認した．

モデル データによって差はあるが，概ね DBVH の方が K の値は少なくなっており，より多

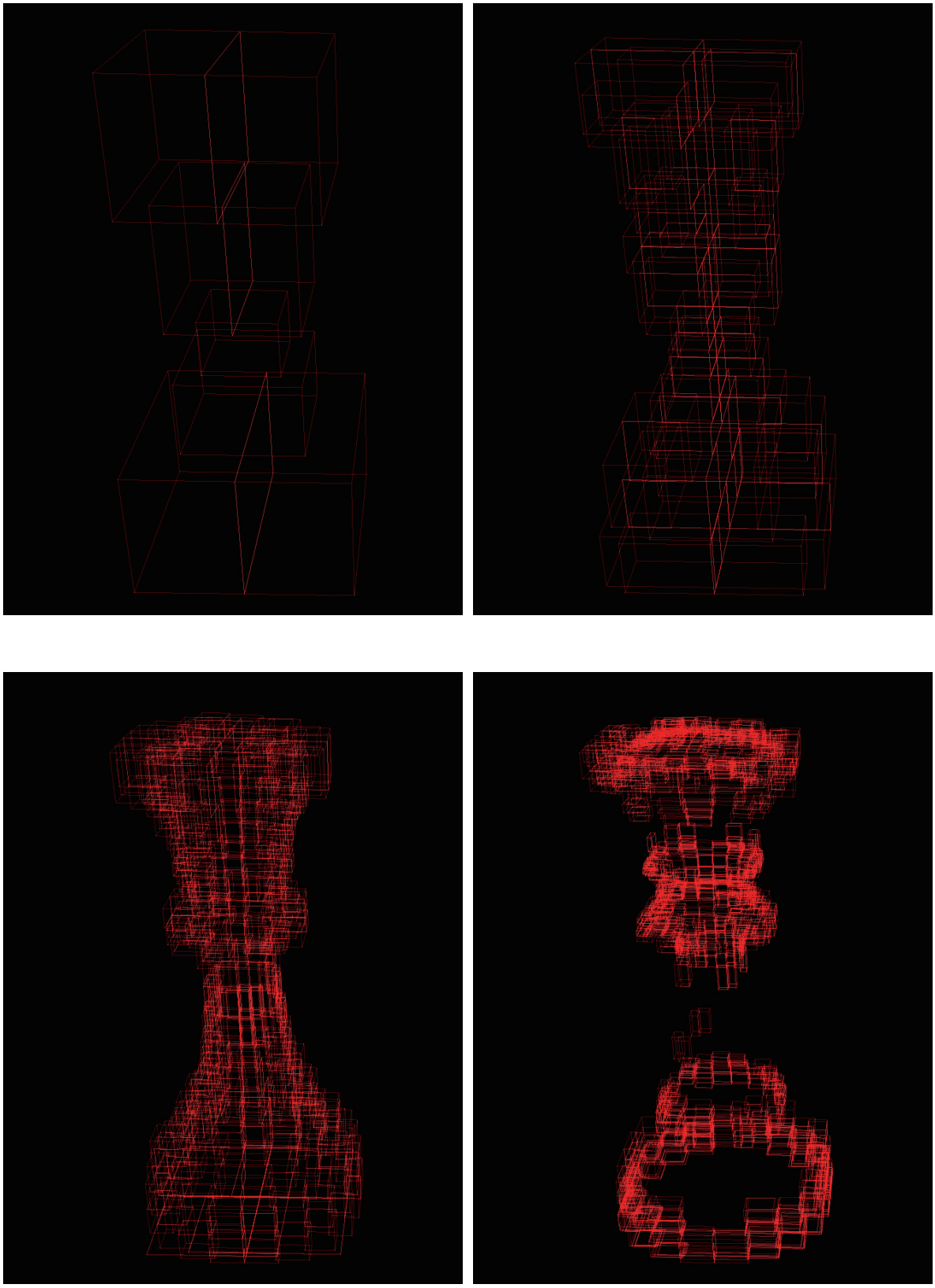


Fig. 6.6　chess モデルに対する MBVH の構築結果．a) 深さ 3 (左上)，b) 深さ 6 (右上)，c) 深さ 9 (左下)，d) 深さ 12 (右下) のノードの AABB を描画した図.

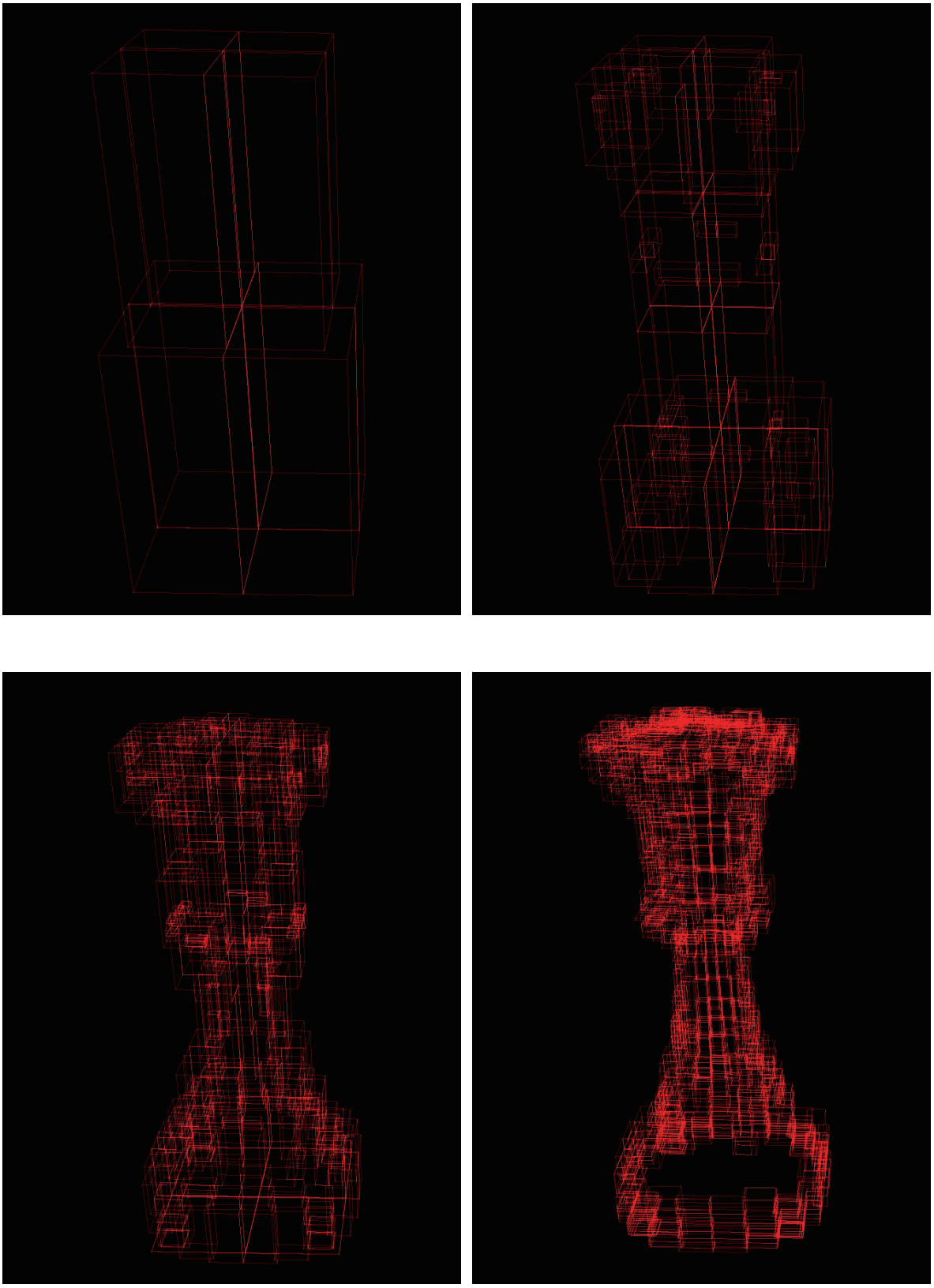


Fig. 6.7　chess モデルに対する LBVH[10] の構築結果．a) 深さ 3 (左上)，b) 深さ 6 (右上)，c) 深さ 9 (左下)，　d) 深さ 12 (右下) のノードの AABB を描画した図.

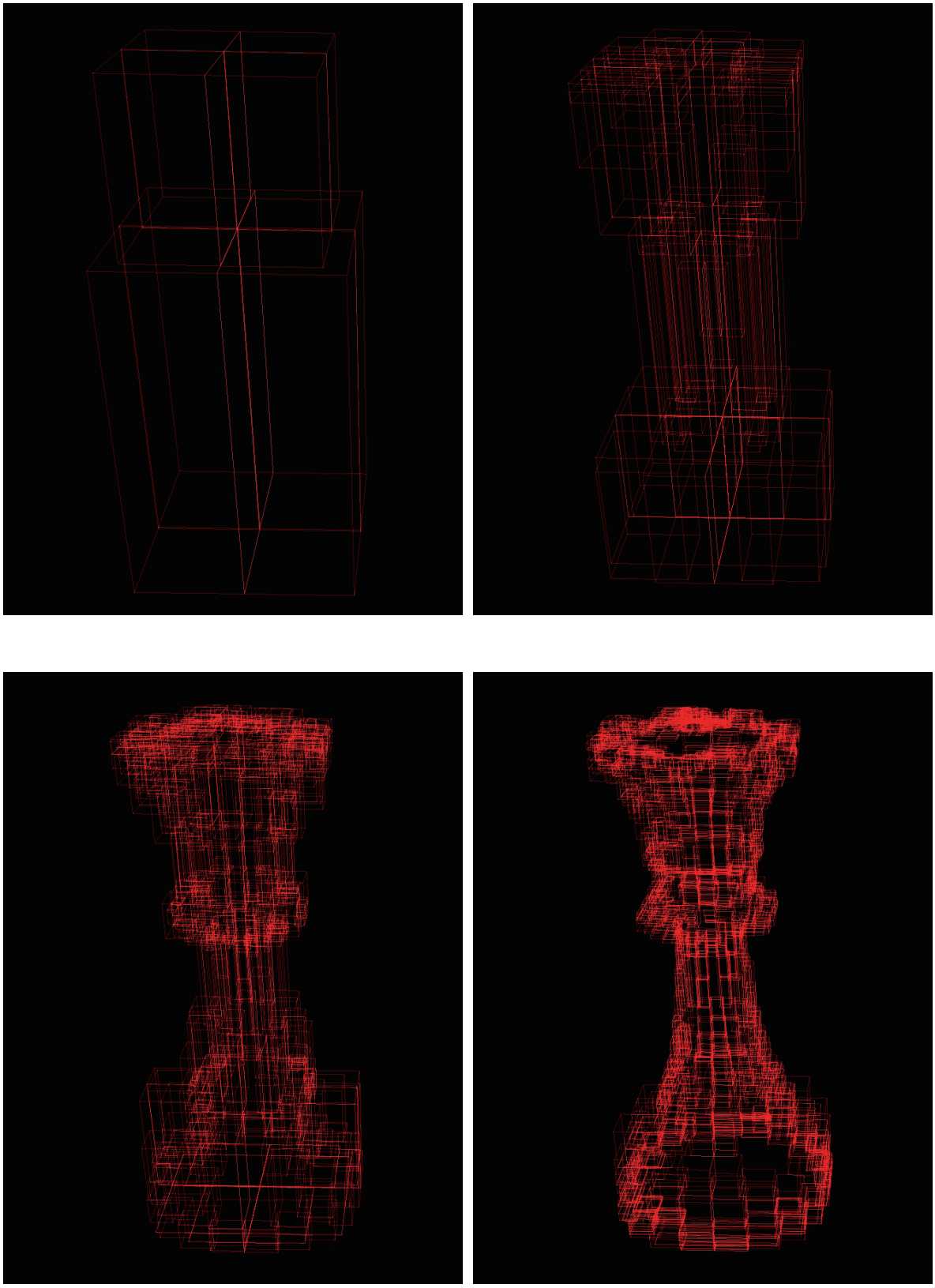


Fig. 6.8　chess モデルに対する MBVH[10,2] の構築結果．a) 深さ 3 (左上)，b) 深さ 6 (右上)，c) 深さ 9 (左下)，　d) 深さ 12 (右下) のノードの AABB を描画した図.

くの primitive をリーフの階層まで分けきることができていることが分かる．ただし，knot のような一様な大きさの primitive で構成されていながら，特殊な primitive の分布をしているものに関しては，逆効果な場合もあることが表から判別できる．そのほか，25 種類程度のモデルデータで比較検証を行ったが，knot 以外に DBVH における K の値が LBVH における値を上回るものは 1 種類のみであり，平均して 20 % 〜 30 % 程度 K の値を低減させることができている．

これは，DBVH がヒストグラムを用いて Morton コードをゆがめることにより，primitive の空間分布を考慮に入れることができていることに起因すると考えられる．

階層構造におけるリーフ ノードの深さについても，その品質の評価指標とすることができる．すべてのリーフ ノードのルートからの深さを測定し，その分布をヒストグラムにしたものが，(Fig.6.9)，(Fig.6.10)，(Fig.6.11)，(Fig.6.12) である，横軸にリーフ ノードのルートからの深さ，縦軸にその深さに分布するリーフ ノードの数をプロットしている．階層構造については，MBVH，LBVH[10]，DBVH[10,2] を採用した．

4 つのモデルすべてにおいて，DBVH は LBVH よりも全体的に浅い階層に分布しており，品質が高い木構造が構成できているといえる．また，buddha モデルおよび uzu モデルにおいては，トップダウンで構築した MBVH よりも浅い階層における分布が確認でき，MBVH よりも品質の BVH が構築できていることが示された．

リーフにおけるノードの深さ分布をカラーチャートを用いて表したものが (Fig.6.13) および (Fig.6.14) である．chess モデル，knot モデル共に，MBVH が青い部分が少なく，一番 BVH の品質が良い．chess モデルにおいては，DBVH[10,2] と LBVH[10] では，DBVH の方が頭の部分などで青い部分が少なく，リーフまでの深さが浅くなっていることが分かる．また，knot モデルにおいても青色の濃い部分が軽減されていることが見て取れる．

以上から，ノードの深さに関する分布より，DBVH における階層構造の品質が LBVH に比べてよくなっていることがわかる．トップダウン構築による手法と比較すると，その品質はまだ劣っているが，シーン形状によっては遜色ない程度まで品質が改善している場合も見受けられる．

6.4　レイ トレーシング法への応用による比較

本研究では，各種の BVH をレイ トレーシング法に応用し，その比較を行った．

本節においては 4.2 節で紹介した，4 種類のヒストグラム歪曲化についても，実装を行い比較を行う．axis delta 法， orthogonal area 法，surface area 法，volume 法の各手法はそれぞ

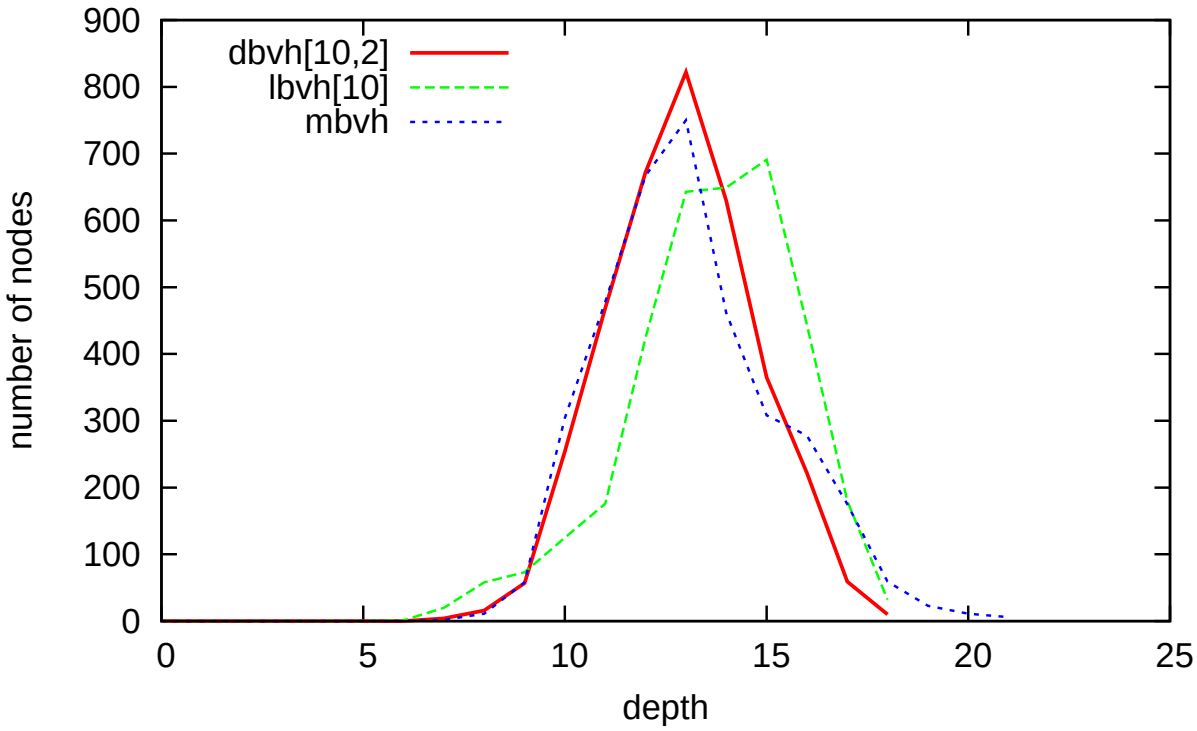


Fig. 6.9　chess モデルにおけるリーフ ノードの深度分布の比較.

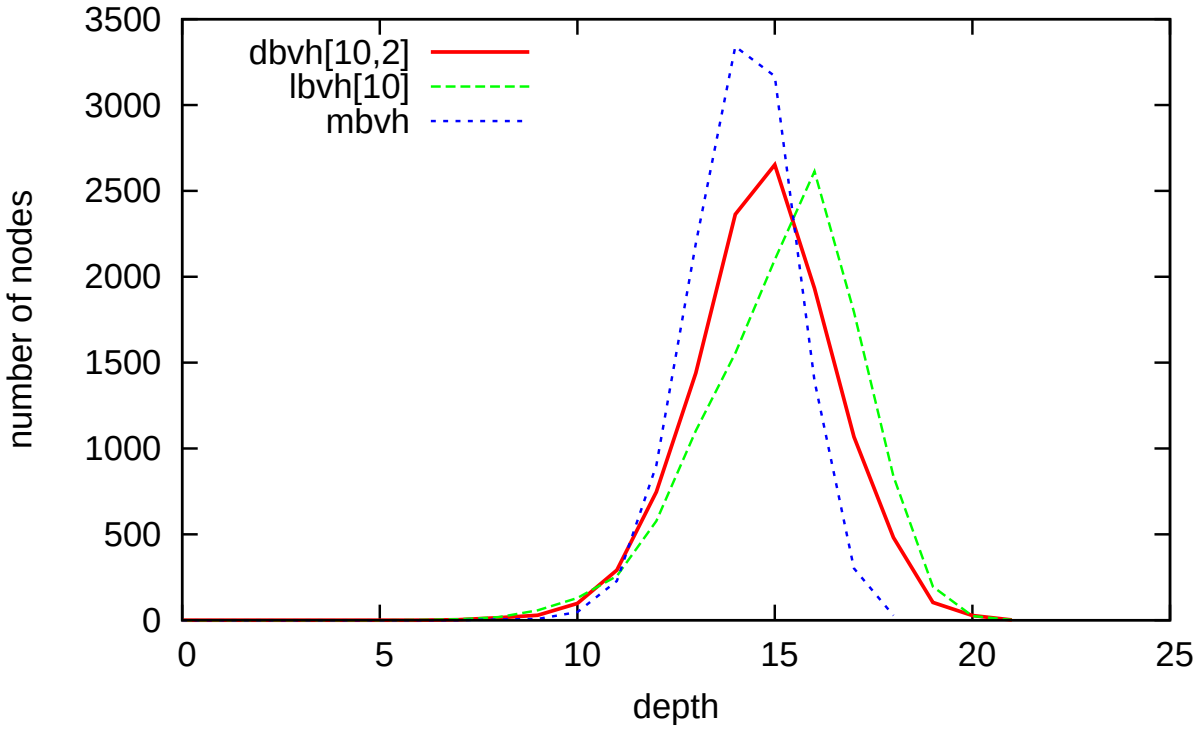


Fig. 6.10　knot モデルにおけるリーフ ノードの深度分布の比較.

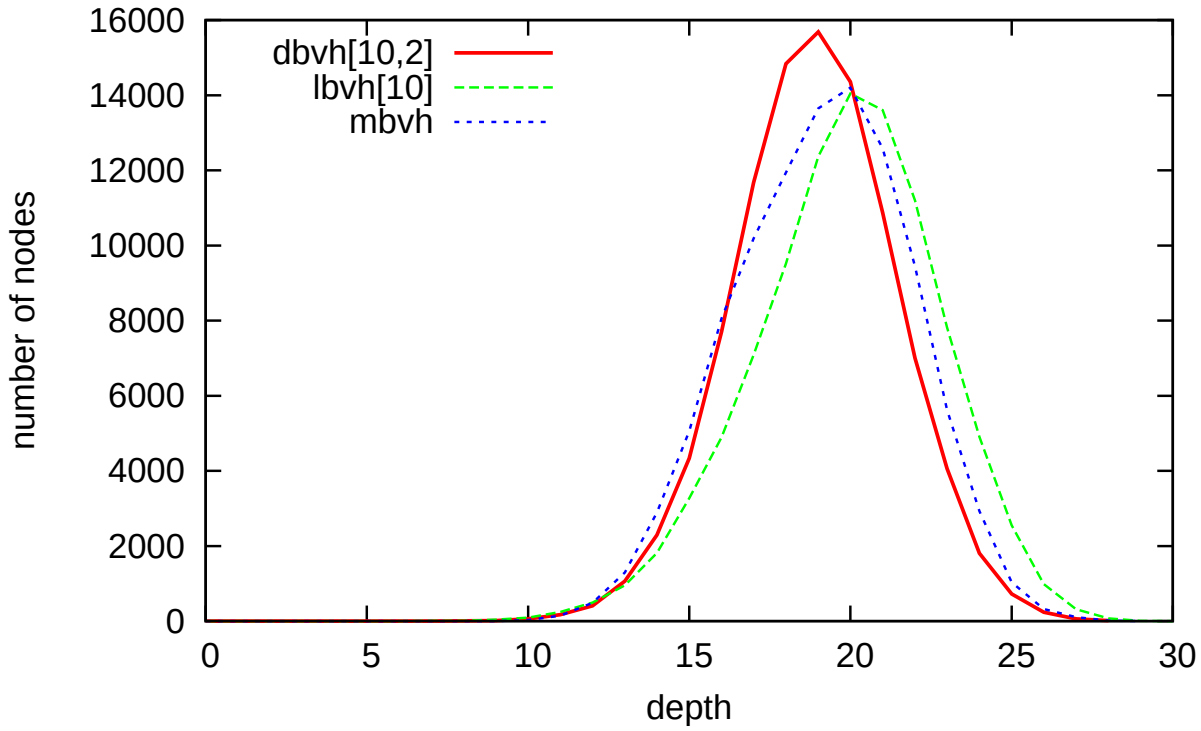


Fig. 6.11　buddha モデルにおけるリーフ ノードの深度分布の比較.

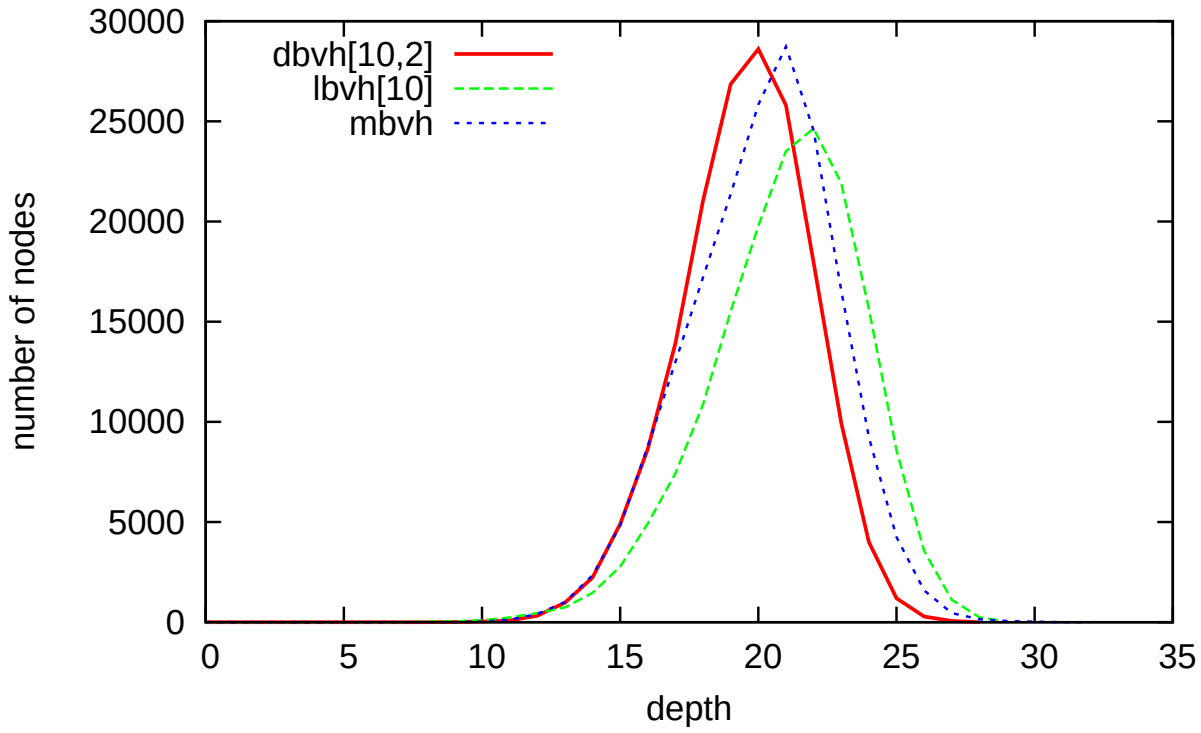


Fig. 6.12　uzu モデルにおけるリーフ ノードの深度分布の比較.

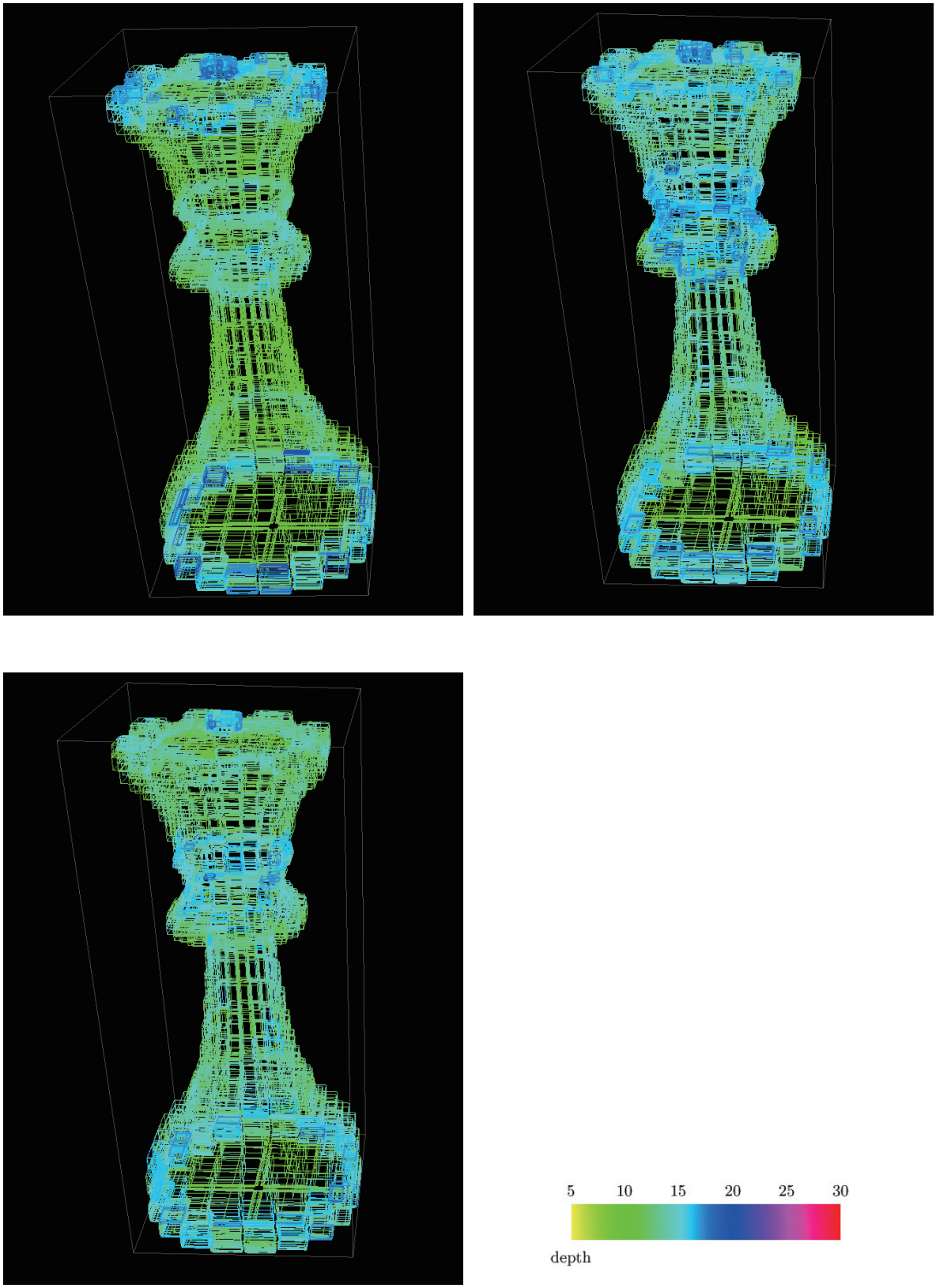


Fig. 6.13　chess モデルに対する MBVH (左上)，LBVH[10] (右上)，DBVH[10,2] (左下) のリーフノードの深さ分布．深さをカラーチャート (右下) を用いて表示した．

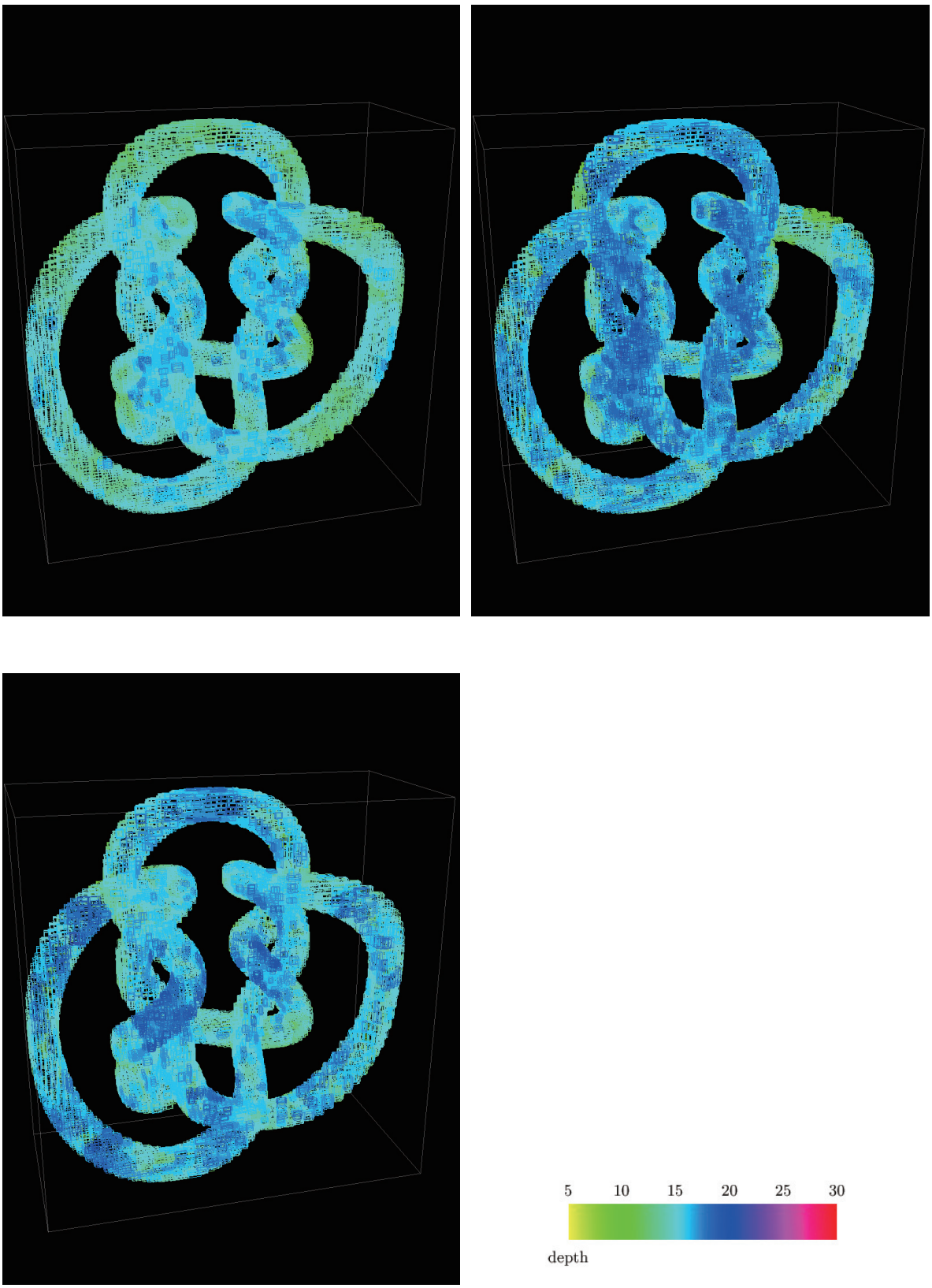


Fig. 6.14　knot モデルに対する MBVH (左上)，LBVH[10] (右上)，DBVH[10,2] (左下) のリーフノードの深さ分布．深さをカラーチャート (右下) を用いて表示した．



Fig. 6.15　左から順に a) chess，b) maxplanck，c) spontza である．

れ-AD，-OA，-SA，-VL の各接尾辞を用いて表記する．

シーンのレンダリング結果画像を (Fig.6.15) に掲載する．モデルデータのポリゴン数等は (Table6.1) に準ずる．なお，chess シーンおよび maxplanck シーンでは，光源として point light を使用し，カメラの位置にセットした．spontza では，合計 12 個の point light を使用している．いずれも解像度は 640 x 640 pixel でレンダリングし，perspective カメラで，1 pixel から 1 ray を飛ばしレンダリングを行った．レンダリング画像では whitted レイ トレーシングを行っている．

これらの 3 つのシーンに対して，シーンの traversal にかかった時間をまとめたグラフが，(Fig.6.16)，(Fig.6.17)，(Fig.6.26)，(Fig.6.27)，(Fig.6.36)，(Fig.6.37) である．それぞれ，chess シーン 1 次光線のみ，chess シーン 1 次光線 + 影光線，maxplanck シーン 1 次光線のみ，maxplanck シーン 1 次光線 + 影光線，spontza シーン 1 次光線のみ，spontza シーン 1 次光線 + 影光線での結果である．いずれも，カメラからの光線の生成に関する時間や BSDF の計算，image plane への投影等の時間は差し引いて記載してあり，逆に影光線の生成や ray-primitive test などの時間は含んだものとなっている．

(Fig.6.16) からわかるとおり，chess シーンで 1 次光線のみの場合，各種の DBVH にくらべて LBVH の効率がわずかに良い．しかし，影光線まで考慮すると，DBVH[10,3] が LBVH をわずかに上回っており，chess シーンにおける DBVH と LBVH の品質に関してほとんど差がないといえる．chess 以外のシーンにおいても，数千〜数万ポリゴンでのシーンにおいては，LBVH と DBVH での品質の差はほとんど出なかった．chess シーンにおいては，4 種類の派生的手法にくらべ，単純に AABB の中心座標の voxel に 1 を加えるヒストグラムの構築法が高品質な BVH を生成している．各 pixel において，走査したノードの数を画像としてマッピングしたものを (Fig.6.18)〜(Fig.6.25) に画像として示す．

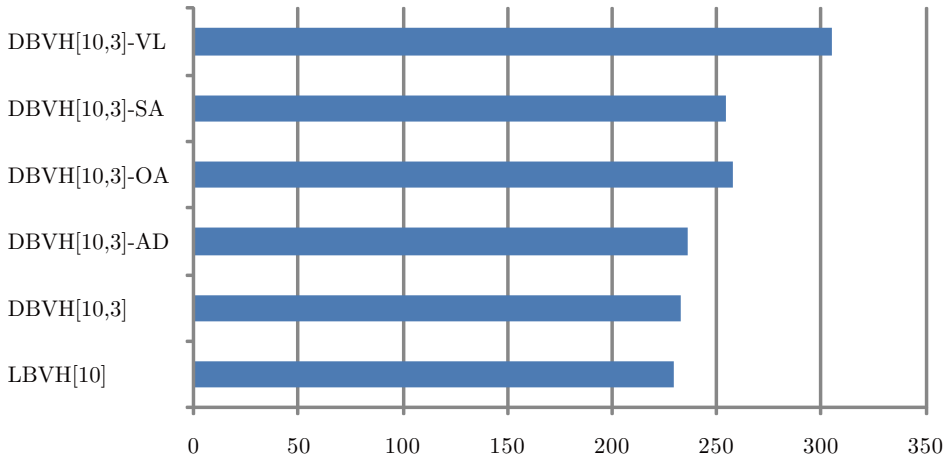


Fig. 6.16　chess シーン (1 次光線のみ) の traversal での時間比較．横軸の単位はミリ秒 (ms) である．

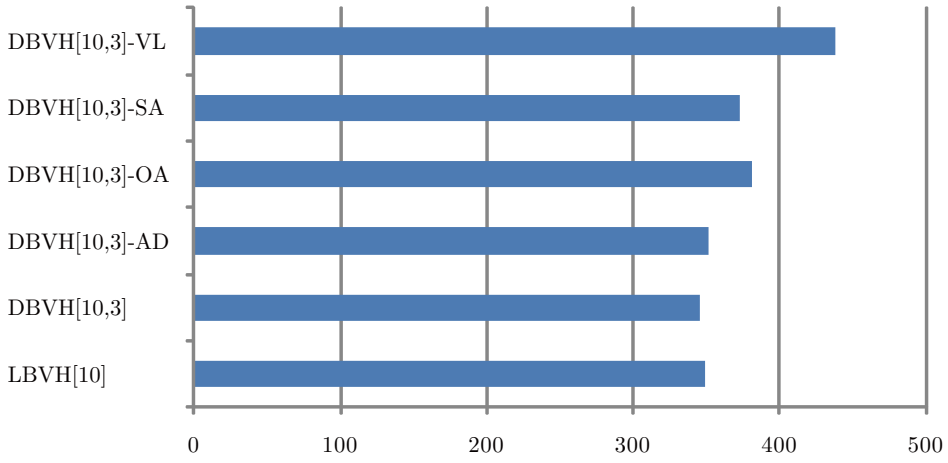


Fig. 6.17　chess シーン (1 次光線 + 影光線) の traversal での時間比較．横軸の単位はミリ秒 (ms) である．

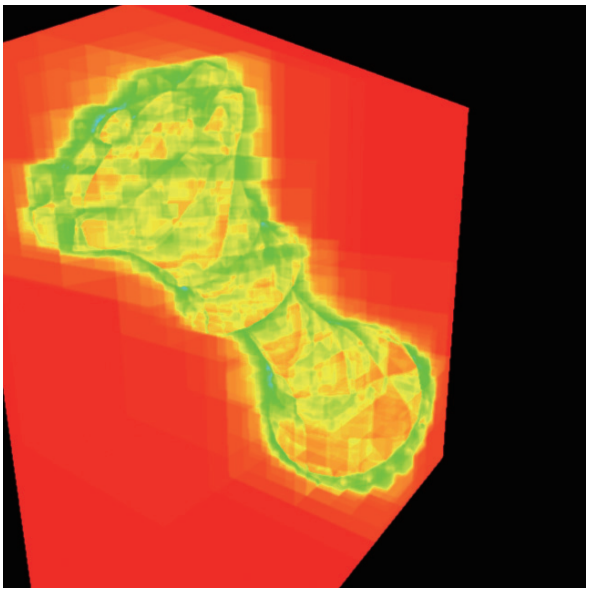


Fig. 6.18　MBVH での走査したノード数の表示 (chess シーン).

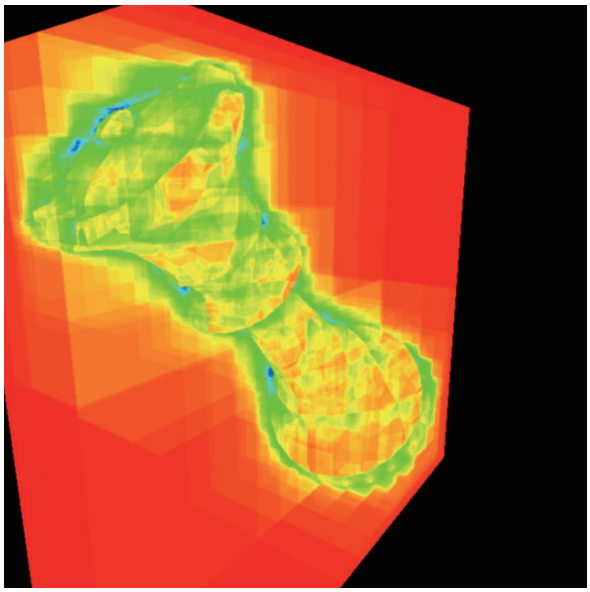


Fig. 6.19　LBVH での走査したノード数の表示 (chess シーン).

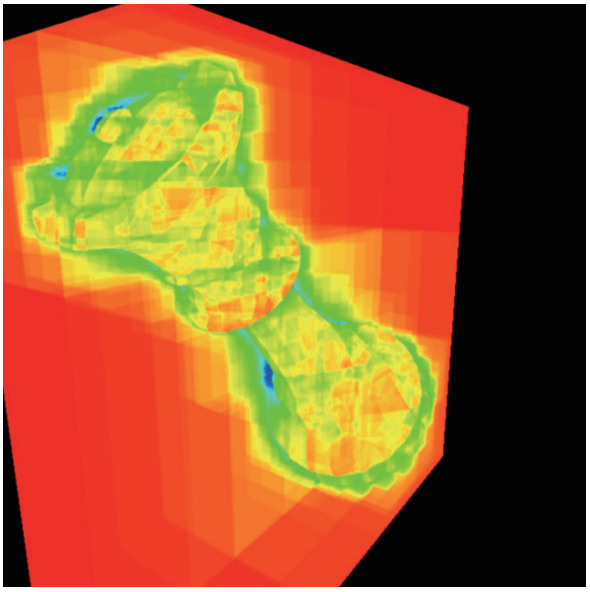


Fig. 6.20　DBVH[10,3] での走査したノード数の表示 (chess シーン).

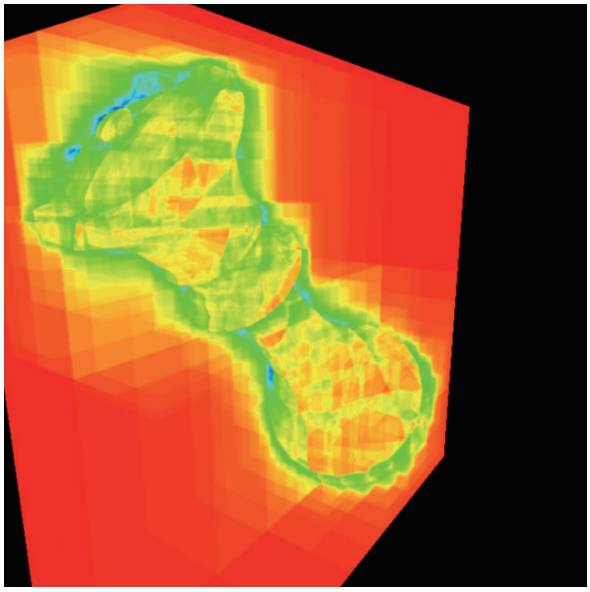


Fig. 6.21　DBVH[10,3]-AD での走査したノード数の表示 (chess シーン).

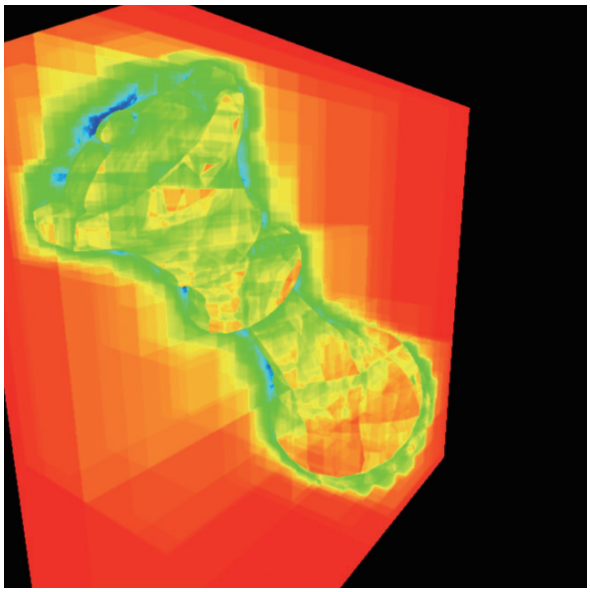


Fig. 6.22　DBVH[10,3]-OA での走査したノード数の表示 (chess シーン).

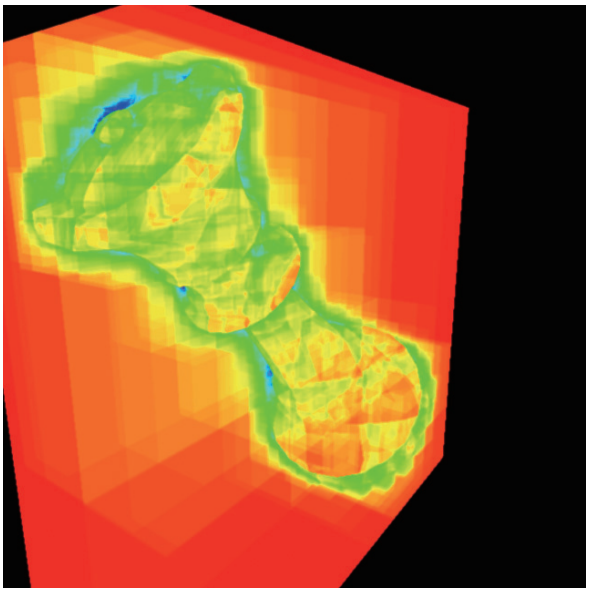


Fig. 6.23　DBVH[10,3]-SA での走査したノード数の表示 (chess シーン).

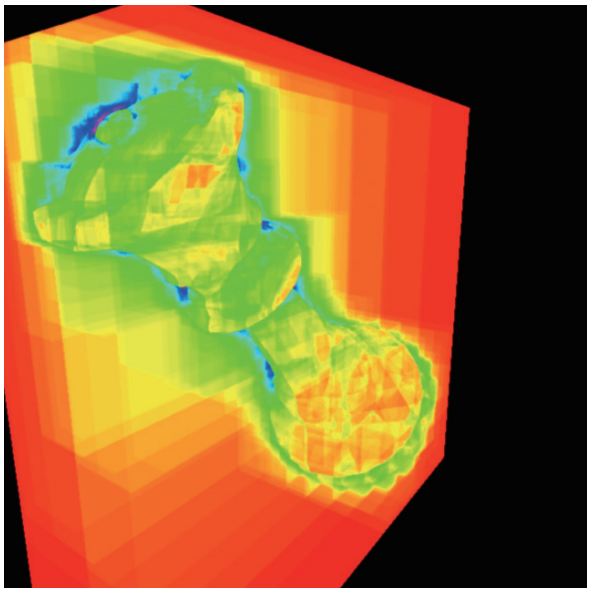


Fig. 6.24　DBVH[10,3]-VL での走査したノード数の表示 (chess シーン).

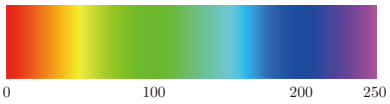


Fig. 6.25　カラーバー. (Fig.6.18)〜(Fig.6.25) で使用.

同様に maxplanck シーンにおける結果 (Fig.6.26) によると，体積を指標として利用してヒストグラムを構成した BVH においてもっともパフォーマンスが優れているという結果が出た．これは，LBVH に比較して約 30 % の高速化を達成している．特に，影光線を考慮した場合は顕著で 40 % 程度の速度向上がみられる．そのほかのヒストグラム歪曲化については，LBVH に劣るものが多いが，影光線を考慮した場合には，surface area 法が LBVH よりも若干効率性で優っている．

chess シーン同様，traversal のノード数を色表示したものを (Fig.6.28)〜(Fig.6.35) に掲載する．

また，非常に primitive が多く，その primitive の大きさの分布にばらつきがあるシーンについても，評価を行った．Spontza Atrium のモデルは GI レンダリングの評価によく用いられるモデルで，その外周を非常に大きな primitive で囲まれている一方で，布のしわやライオン像などの緻密な部分には非常に小さな primitive が多用されている．そのため，空間データ構造の品質の維持が難しいモデルデータであるといえる．

spontza における速度の計測結果を (Fig.6.36) 及び (Fig.6.37) に掲載する．

spontza においては，orthogonal area 法の DBVH が一番品質が良いという結果が得られた．ついで，LBVH でのレンダリング速度が僅差で高速である．なお，MBVH については，影光線なしで 4 秒以上，影光線ありでは 40 秒以上レンダリングにかかっており，極度に BVH の品質が低下してしまった．これは，(Fig.6.38) をみてわかるとおり床面の階層が深くなってしまっていることに起因する．このシーンでは，そのほとんどの光線が床と初めに交差するため，床面の階層が深いことがシーンのレンダリング時間を致命的に遅くする原因となっている．このように，空間分割構造はシーンのなかでどこをレンダリングするかによっても，その性能が変わってしまうため，非常に評価が難しいといえる．

(Fig.6.38)〜(Fig.6.45) に spontza での traversal 数の可視化を掲載する．chess シーンや maxplanck シーンと異なり，手法によってその traversal 数の分布が大きく異なるという結果となった．

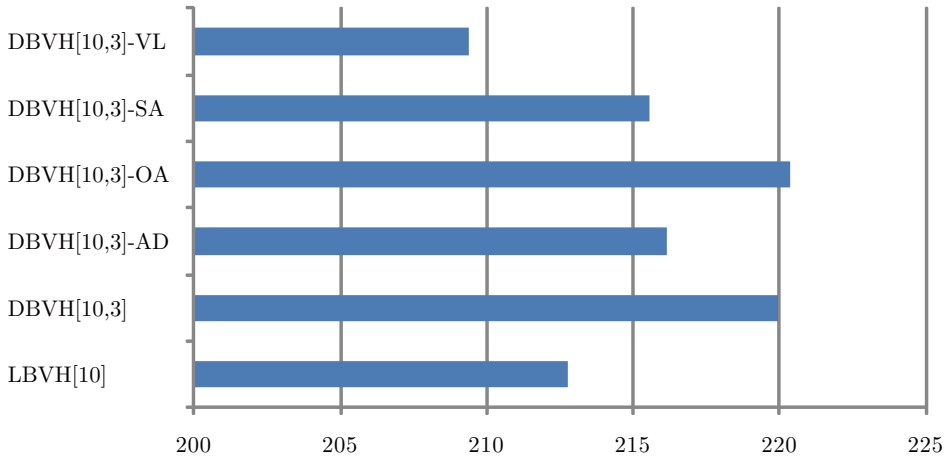


Fig. 6.26　maxplanck シーン (1 次光線のみ) の traversal での時間比較．横軸の単位はミリ秒 (ms) である．

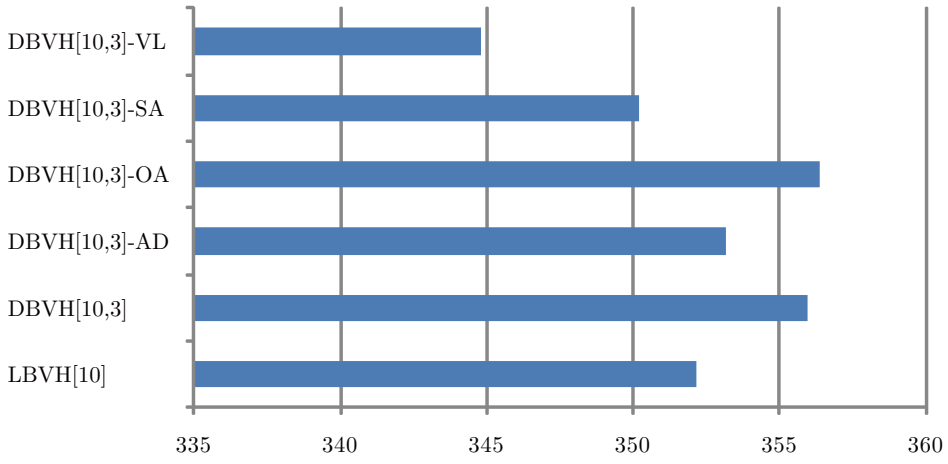


Fig. 6.27　maxplanck シーン (1 次光線 + 影光線) の traversal での時間比較．横軸の単位はミリ秒 (ms) である．

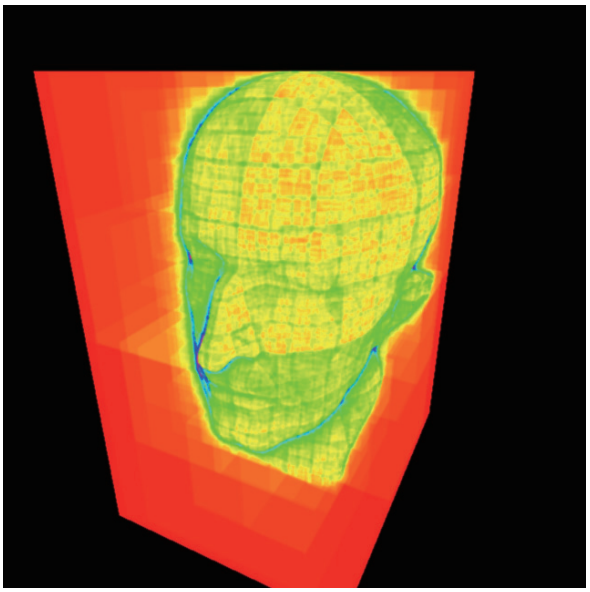


Fig. 6.28　MBVH での走査したノード数の表示 (maxplanck シーン).

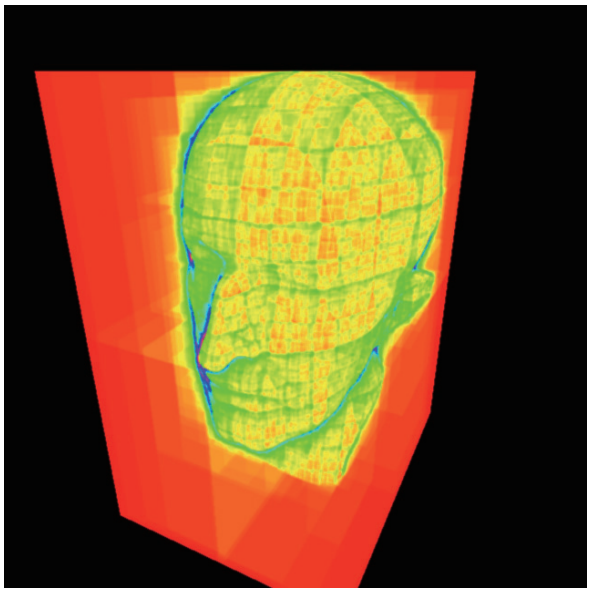


Fig. 6.29　LBVH での走査したノード数の表示 (maxplanck シーン).

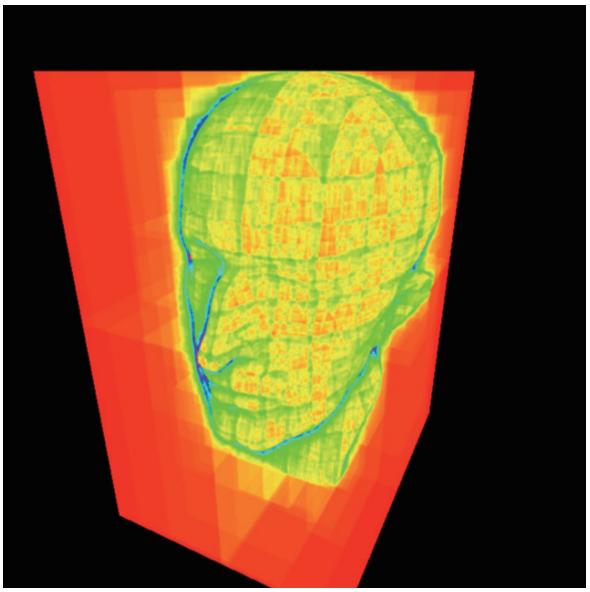


Fig. 6.30　DBVH[10,3] での走査したノード数の表示 (maxplanck シーン).

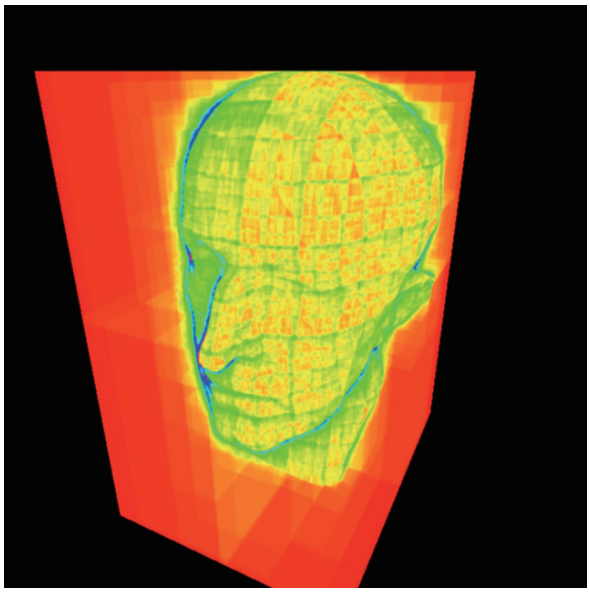


Fig. 6.31　DBVH[10,3]-AD での走査したノード数の表示 (maxplanck シーン).

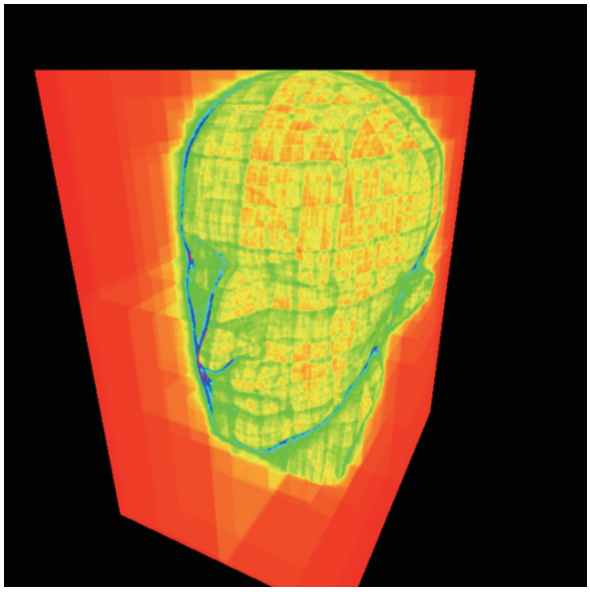


Fig. 6.32　DBVH[10,3]-OA での走査したノード数の表示 (maxplanck シーン).

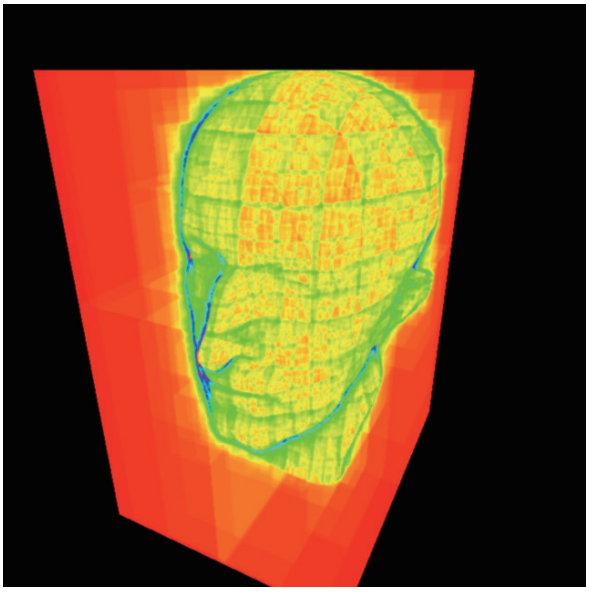


Fig. 6.33　DBVH[10,3]-SA での走査したノード数の表示 (maxplanck シーン).

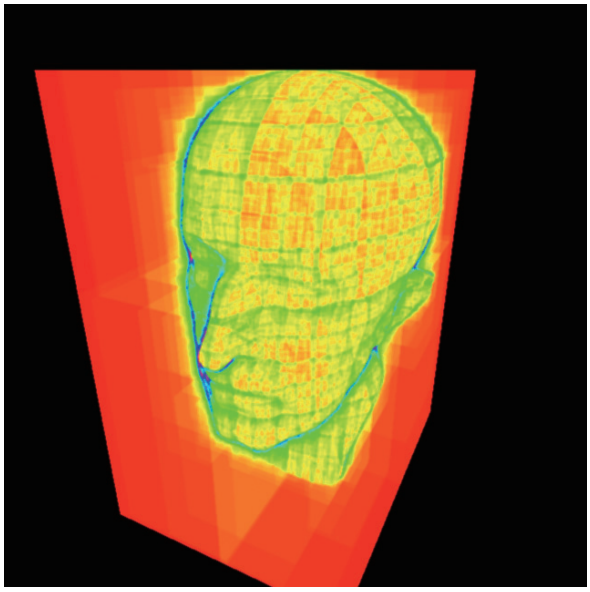


Fig. 6.34　DBVH[10,3]-VL での走査したノード数の表示 (maxplanck シーン).

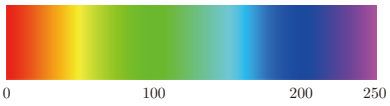


Fig. 6.35　カラーバー. (Fig.6.28)〜(Fig.6.35) で使用.

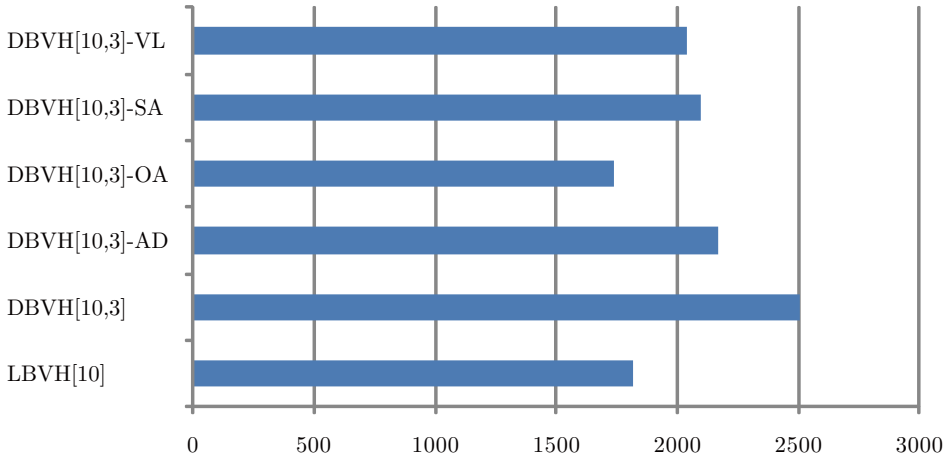


Fig. 6.36　spontza シーン (1 次光線のみ) の traversal での時間比較．横軸の単位はミリ秒 (ms) である．

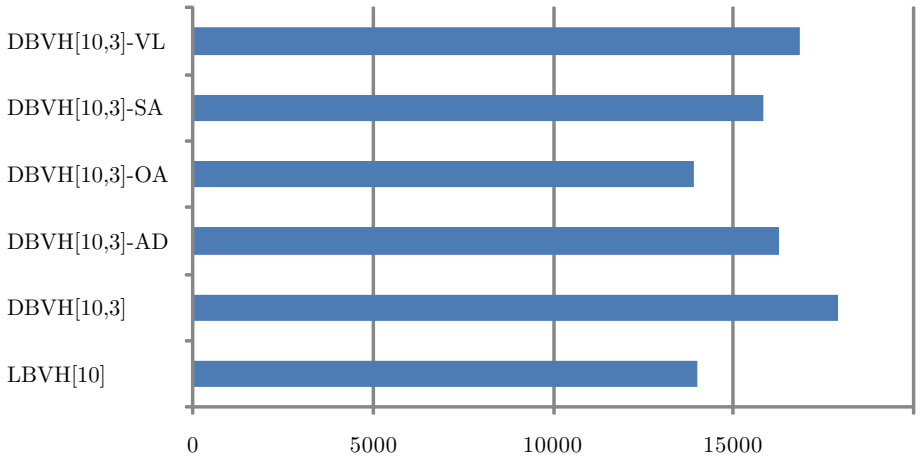


Fig. 6.37　spontza シーン (1 次光線 + 影光線) の traversal での時間比較．横軸の単位はミリ秒 (ms) である．

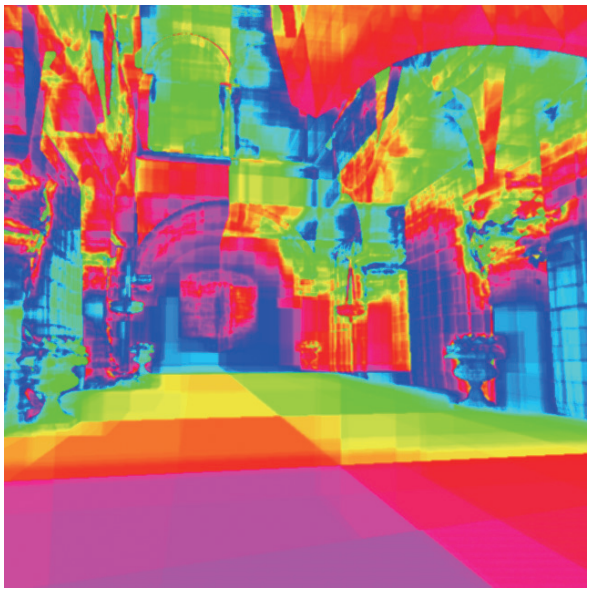


Fig. 6.38　MBVH での走査したノード数の表示 (spontza シーン).

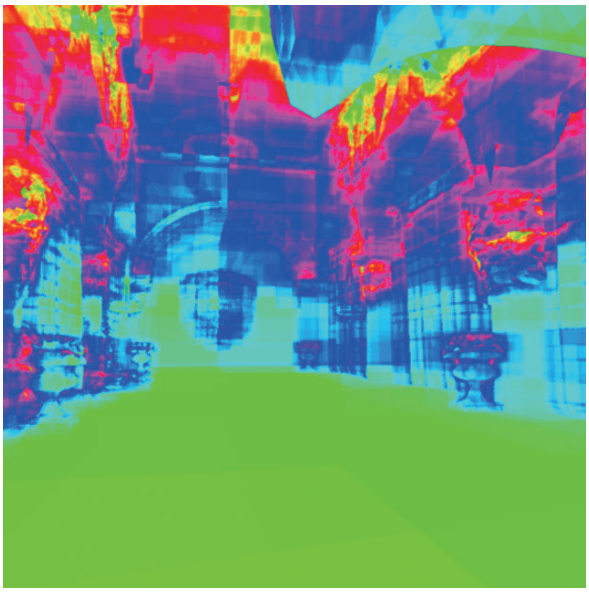


Fig. 6.39　LBVH での走査したノード数の表示 (spontza シーン).

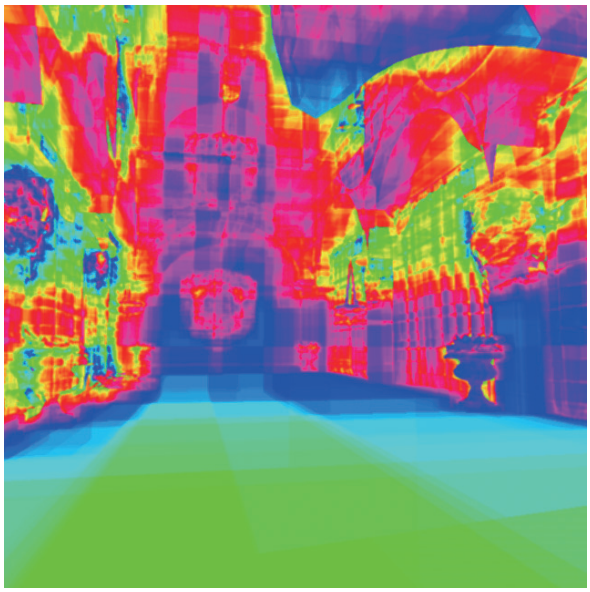


Fig. 6.40　DBVH[10,3] での走査したノード数の表示 (spontza シーン).

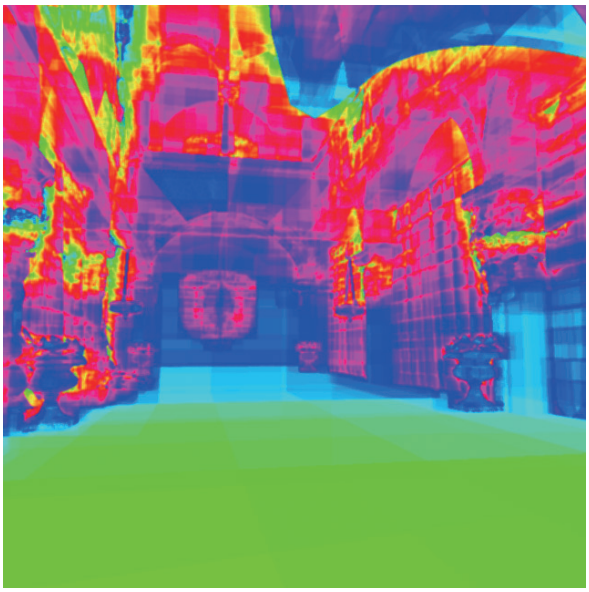


Fig. 6.41　DBVH[10,3]-AD での走査したノード数の表示 (spontza シーン).

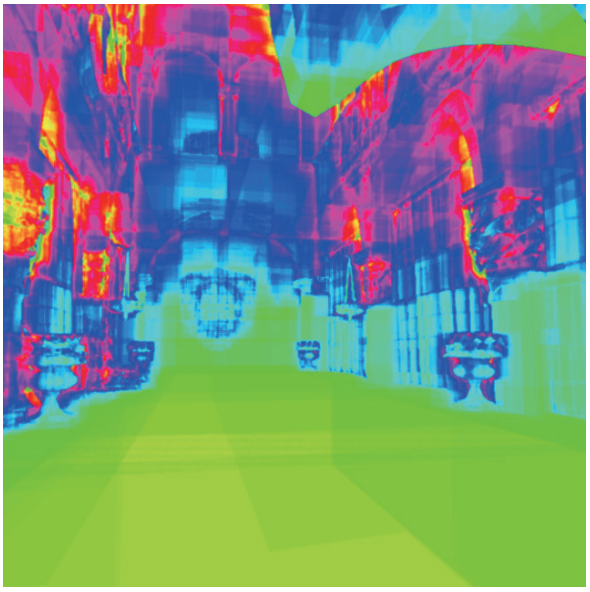


Fig. 6.42　DBVH[10,3]-OA での走査したノード数の表示 (spontza シーン).

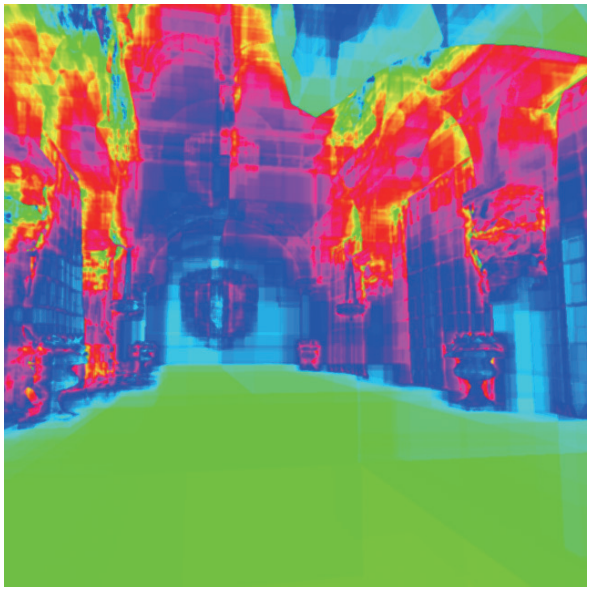


Fig. 6.43　DBVH[10,3]-SA での走査したノード数の表示 (spontza シーン).

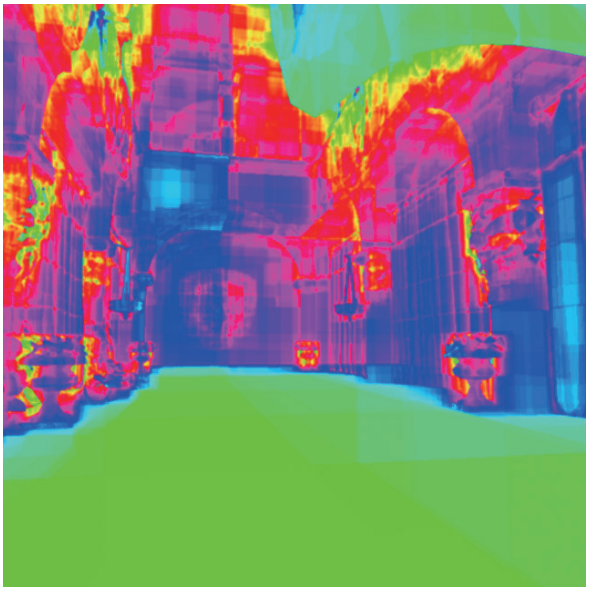


Fig. 6.44　DBVH[10,3]-VL での走査したノード数の表示 (spontza シーン).

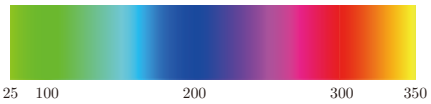


Fig. 6.45　カラーバー. (Fig.6.38)〜(Fig.6.45) で使用.

第 7 章

結論

7.1 結論

本研究においては，新たな階層的空間分割手法である，DBVH の提案を行った．
本手法の特徴は以下の 4 点である．

1. 階層的空間データ構造の中で AABB を BV とする BVH である．
2. データ並列アルゴリズムでの実装に適している．
3. 構築にあたってヒストグラムを用い，空間的局所性を考慮にいれている．
4. ヒストグラムの構成法に関して，複数の戦略を選択することができる．

実際に，提案手法の実装を行い，レイ トレーシング法への適応し，その効率性および構築速度を比較した．

提案手法である DBVH は，LBVH に比べて 10 % 程度のオーバーヘッドで，木構造の比較では MBVH に匹敵する品質を得ることができた．

また，レイ トレーシング法における応用では，LBVH および DBVH の効率はシーンに依るところが大きく，最適なヒストグラムの構成法を選択することで最大 40 % 程の効率性向上が図ることができるという結果となった．

7.2 今後の展望

本研究における空間分割手法にはレイ トレーシング法以外にも様々な応用が考えられる．衝突判定や KNNS，物理シミュレーションなどはその代表例である．それらのアプリケーションへの応用を行い，さらにその効率性・適合性を検証する必要がある．

また，ヒストグラムの構築においては，複数の手法を提示したが，シーンによって，どの手法が優れているかが異なるということが明らかになった．これには，更なる検証とどのようなシーンにおいても一定の品質以上の BVH を構築することのできる手法の提案が待たれる．どのような戦略で構築するかについては，アプリケーションによる比較も重要であると考えられるため，さらなる研究の余地が残る．

7.3 あとがき

本論文では，リアルタイム レイ トレーシング法への応用に向けて，空間分割に関して議論を行った．

近年では，計算機の性能の向上とともに，ハードウェア ラスタライゼーションによるレンダリング手法から，ソフトウェア レイ トレーシング法へと回帰する流れが見受けられる．これから先，様々な手法の登場によりラスタライゼーションとレイ トレーシング法の境はなくなっていくであろう．それに際して，空間の表現は一番に課題になる部分である．本手法ではまだまだ完璧とはいえないが，更なる改良を行い，より質の高い映像表現が手軽に実現できるようになることを願ってやまない．

また，高度な映像表現においては，レンダリングの質に対して，その動きや微細構造の表現の質が劣っていることが，現場レベルで指摘されている．物理シミュレーションによる手法はもちろん，生物的本質をとらえたモデリング手法や心理的効果を考慮に入れたアニメーション制作手法など，まだまだ発展の余地の残された分野である．本研究では，極めて基礎的かつ汎用的な題材を扱ったが，実際の応用を念頭に置いた，実用性に主眼をおいて論を進めてきたつもりである．これらの新たな可能性を含む分野においても，本手法に代表されるような，効率的な空間分割手法を利用することで，インタラクティブな応用ができるようになるであろう．

謝辞

本論文を書くにあたって，様々な方の多大なるご指導やご協力，支えのおかげで無事完成にこぎつけることができました．この場を借りまして，各位に深くお礼申し上げたいと思います．

研究室の教授として，卒業論文の頃から東京大学大学院情報学環 教授 河口洋一郎氏にお世話になりました．アーティスティックな感性や無尽蔵に湧き出る発想力は，日頃の研究に於いて何よりの刺激となりました．ギリギリのタイミングでの研究テーマ変更の申し出にあたっても，その決断を快諾していただき，非常に助かりました．また，レンダリング画像の生成にあたっては，一部に氏デザインのモデルデータを使用させていただきました．

本研究にあたって，Havok 原田隆宏氏には非常に熱心なご指導を賜りました．修士論文のテーマが行き詰まっている際に，5 時間もの長きに及ぶテーマ選びに付き合っていただきました．その後も，アイルランドでの業務でお忙しいにも関わらず，長文のメールを即座に返してくださるなど，親身になって相談に乗ってくださり，氏の尽力なしでは本研究は成し得ることが出来ませんでした．深謝するとともに，今後の研究活動に於いても，尊敬すべき目標として練磨して参りたいと思います．

東京大学大学院情報学環 助教 米倉将吾氏とは，研究会等で熱い議論を交わし，本研究の意義や題目，論文構成などを考える上で非常に貴重なアドバイスを頂きました．他研究分野の研究者ではありますが，明確なビジョンを元に issue base で行う，氏の研究の描き方は，私の研究に対する姿勢を形作る上での模範となりました．また，予算管理やサーバー管理など，研究生活に欠かせない仕事を一手に担っていただいておりましたことにも深く感謝致します．

アルバイト業務での上司である，プロメテック・ソフトウェア株式会社 CTO 政家一誠氏には，寝坊や遅刻などご迷惑をかけることも多くありましたが，それでも優しく接していただき，本研究にあたる，身近な相談相手として研究期間中大変お世話になりました．原田氏とともにテーマ選びにもご参加くださり，貴重なアドバイスを頂きました．delegate の使い方や SSE の命令，CUDA でのエラー解決など，実装面をはじめ，氏から学んだことは枚挙にいと

まがありません．今後も関わる機会も多いかと思いますが，変わらぬご鞭撻のほどよろしくお願いします．

LAAS- CNRS 多田隈理一郎氏には河口研究室 講師 在任中に隣席でよく相談させて頂きました．短い期間ながら，深夜に我々の意見や主張に耳を傾けていただき，アドバイスしてくださったことが何度もありました．

堀聖司氏には研究室の運営にあたって，展示活動や講演準備など常に忙しく働きまわっているにもかかわらず，その陽気な人柄で我々の笑いを誘い，和ませてくれていただきました．コンテンツ産学連携プログラムからの物品借用や，研究機材購入に関する相談をさせて頂いたり，映像制作における実務的なお話をしてくださったりと研究室での研究活動や制作活動を力強くバックアップしてくださいました．

研究室事務員の小菅瑞恵氏には，機材の注文・借用やソフトウェアの発注など事務的な面で大変お世話になりました．印刷用紙やトナーなど消耗品を大量に消費するにもかかわらず，常に切らすことなく補充してくださいました．

本研究にあたっては，板橋さやか氏のモデリングしたデータを使用させて頂きました．特に，アンラッキーやキレイハナなどのモデルは研究室でのアプリケーションの標準データとして多く使用させていただきました．また，ラッキーのデスクトップ画像を作成して頂いたり，花火や肉の会などのイベントを企画して頂いたりと，氏のおかげで楽しい研究を送る事ができました．

李ジュヒョン氏，岡部理加氏並びに上田あい氏には ASIAGRAPH や湯島聖堂での展示など，対外的なイベントに際し特にお世話になりました．迅速かつ適切な仕事のおかげで，シフトや展示準備などをスムーズにこなすことができ，研究活動へ専心することができました．上田氏にはリアルタイム デモ用のクラゲモデルを提供していただきました。

学際情報学府博士課程 鶴岡修平氏には河口研究室における CG 系の先輩としてお世話になりました．本研究でも利用しているライブラリの構築にも携わっていただいていました．

安田廉氏は学部時代から，行動を共にすることが多く，論文講読会の実施による知識の研磨や，普段の研究における議論など，互いに切磋琢磨してきました．本研究においても，2 人でホワイト ボードを使ってブレイン ストーミングを行い，テーマ探しを行うことから始めました．同じ CG 系の研究者として大変貴重な存在であり，今後も互いに研鑽していければと思います．

豊泉徹平氏はロボット系の研究者でありながら，研究に関する議論を交わすことも多くありました．その多くはボケの応酬で，とりとめのないことが多かったのは事実ですが，その中か

ら生まれる自由な発想は今後も大切にしていきたいものの一つです．本論文執筆中に，真夜中に「おはようございまーす」と神出鬼没に現れる氏を見かける機会も多く，夜型での研究における同志でありました．

Sejuti Rahman 氏とは私のつたない英語のせいでなかなかコミュニケーションが取れなかった面もありましたが，連続体の物理シミュレーションという近い分野での研究者として，積極的な議論を行いました．英語の校正などお世話になることも多くありました．

石井達也氏，市橋茂氏，松原玄彦氏，巽淳一氏，棚橋譲氏，長澤康平氏，中川佑輔氏には河口研究室の後輩として，ネットワークや計算機の管理を引き継いでいただいたり，論文講読会で論文を発表していただいたりと，研究室での研究活動の活性化にも一役かっていただきました．自主性が重んじられ，ワンマンプレーに走ることの多い河口研究室ですが，広い視点を持ち，有機的に物事を考える上で，研究室会の開催や講読会の実施は非常に有意義な取り組みであったと思います．これからも続けていくとともに，お互いによい研究生活を送っていければと思います．

Twitter 上で論文の進み具合を報告し，励まし合うなど共に戦った仲間として制作展の仲間や学際情報学府の同期メンバー，SQUARE ENIX の内定者の皆様方には感謝に耐えません．特に，学際情報学府 池内研究室修士 工藤雷太氏は河口研究室に来ることも多く，研究その他の話題で大いに盛り上がりました．

今年の年末年始は実家への帰省がかないませんでしたが，最後に遠くから支えてくださっている家族に感謝したいと思います．

修士生活を送るにあたって，ここには書ききれないほど多くの方々に支えられて過ごすことができました．皆様，本当にありがとうございました．

参考文献

[1] A. M. Andrew. Another efficient algorithm for convex hulls in two dimensions. *Information Processing Letters*, Vol. 9, No. 5, pp. 216 – 219, 1979.

[2] Arthur Appel. Some techniques for shading machine renderings of solids. In *AFIPS '68 (Spring): Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pp. 37–45, New York, NY, USA, 1968. ACM.

[3] James Arvo and David Kirk. Fast ray tracing by ray classification. In *SIGGRAPH '87: Proceedings of the 14th annual conference on Computer graphics and interactive techniques*, pp. 55–64, New York, NY, USA, 1987. ACM.

[4] James Arvo and David Kirk. A survey of ray tracing acceleration techniques. pp. 201–262, 1989.

[5] S. Bandi and D. Thalmann. An adaptive spatial subdivision of the object space for fast collision detection of animated rigid bodies. In *Proc. Computer Graphics Forum*, Vol. 14, pp. 259–70, 1995. Comput. Graphics Lab., Swiss Federal Inst. of Technol., Lausanne, Switzerland.

[6] David Baraff. *Dynamic simulation of nonpenetrating rigid bodies*. PhD thesis, Ithaca, NY, USA, 1992.

[7] C. Bradford Barber, David P. Dobkin, and Hannu Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Trans. Math. Softw.*, Vol. 22, No. 4, pp. 469–483, 1996.

[8] Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, and Bernhard Seeger. The r*-tree: an efficient and robust access method for points and rectangles. *SIGMOD Rec.*, Vol. 19, No. 2, pp. 322–331, 1990.

[9] David Benson and Joel Davis. Octree textures. *ACM Trans. Graph.*, Vol. 21, No. 3, pp. 785–790, 2002.

[10] Jon Louis Bentley. Multidimensional binary search trees used for associative searching.

Commun. ACM, Vol. 18, No. 9, pp. 509–517, 1975.

[11] Jon Louis Bentley. K-d trees for semidynamic point sets. In *SCG '90: Proceedings of the sixth annual symposium on Computational geometry*, pp. 187–197, New York, NY, USA, 1990. ACM.

[12] A. Bowyer. Computing Dirichlet tessellations. *The Computer Journal*, Vol. 24, No. 2, pp. 162–166, 1981.

[13] Donald R. Chand and Sham S. Kapur. An algorithm for convex polytopes. *J. ACM*, Vol. 17, No. 1, pp. 78–86, 1970.

[14] David Cline, Justin Talbot, and Parris Egbert. Energy redistribution path tracing. *ACM Trans. Graph.*, Vol. 24, No. 3, pp. 1186–1195, 2005.

[15] Michael F. Cohen, John Wallace, and Pat Hanrahan. *Radiosity and realistic image synthesis*. Academic Press Professional, Inc., San Diego, CA, USA, 1993.

[16] Robert L. Cook, Thomas Porter, and Loren Carpenter. Distributed ray tracing. *SIGGRAPH Comput. Graph.*, Vol. 18, No. 3, pp. 137–145, 1984.

[17] K. Crane, I. Llamas, and S. Tariq. Real-time simulation and rendering of3d fluids. In Hubert Nguyen, editor, *GPU Gems 3*, chapter 30, pp. pp.633–675. NVIDIA, 2007.

[18] Carsten Dachsbacher, Christian Vogelgsang, and Marc Stamminger. Sequential point trees. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pp. 657–662, New York, NY, USA, 2003. ACM.

[19] Dammertz, H., Hanika, J., Keller, and A. Shallow bounding volume hierarchies for fast simd ray tracing of incoherent rays. *Computer Graphics Forum*, Vol. 27, No. 4, pp. 1225–1233, June 2008.

[20] M. de Berg. Linear size binary space partitions for uncluttered scenes. *Algorithmica*, Vol. 28, No. 3, pp. 353–366, 12 2000.

[21] Mark de Berg, Marc van Kreveld, Mark Overmars, and Otfried Schwarzkopf. *Computational geometry: algorithms and applications*, chapter 1, pp. 2–8. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 1997.

[22] Matthew Dickerson, Christian A. Duncan, and Michael T. Goodrich. K-d trees are better when cut on the longest side. In *Algorithms - ESA 2000*, pp. 179–190. Springer Berlin / Heidelberg, 2000.

[23] William F. Eddy. A new convex hull algorithm for planar sets. *ACM Trans. Math. Softw.*,

Vol. 3, No. 4, pp. 398–403, 1977.

[24] Stephen A. Ehmann and Ming C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. In *in Computer Graphics Forum*, pp. 500–510, 2001.

[25] Tim Foley and Jeremy Sugerman. Kd-tree acceleration structures for a gpu raytracer. In *HWWS '05: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pp. 15–22, New York, NY, USA, 2005. ACM.

[26] A. R. Forrest. Computational Geometry. *Proceedings of the Royal Society of London. A. Mathematical and Physical Sciences*, Vol. 321, No. 1545, pp. 187–195, 1971.

[27] S Fortune. A sweepline algorithm for voronoi diagrams. In *SCG '86: Proceedings of the second annual symposium on Computational geometry*, pp. 313–322, New York, NY, USA, 1986. ACM.

[28] Henry Fuchs, Zvi M. Kedem, and Bruce F. Naylor. On visible surface generation by a priori tree structures. In *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pp. 124–133, New York, NY, USA, 1980. ACM.

[29] A. Fujimoto, T. Tanaka, and K. Iwata. Arts: Accelerated ray-tracing system. *Computer Graphics and Applications, IEEE*, Vol. 6, No. 4, pp. 16–26, April 1986.

[30] Yasutaka Furukawa, Brian Curless, Steven M. Seitz, and Richard Szeliski. Reconstructing building interiors from images. In *ICCV 2009 (oral)*, 2009.

[31] P. L. George and H. Borouchakis. *Delaunay Triangulation and Meshing: Application to Finite Elements*. Hermes, France, 1998.

[32] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi. A fast procedure for computing the distance between complex objects in three-dimensional space. *Robotics and Automation, IEEE Journal of*, Vol. 4, No. 2, pp. 193–203, 1988.

[33] Jeffrey Goldsmith and John Salmon. Automatic creation of object hierarchies for ray tracing. *IEEE Comput. Graph. Appl.*, Vol. 7, No. 5, pp. 14–20, 1987.

[34] R.A. Goldstein and Nagel. R. 3-d visual simulation. *Simulation*, Vol. 16, No. 1, pp. 25–31, 1971.

[35] Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. *SIGGRAPH Comput. Graph.*, Vol. 18, No. 3, pp. 213–222, 1984.

[36] Dan Gordon and Shuhong Chen. Front-to-back display of bsp trees. *IEEE Comput. Graph. Appl.*, Vol. 11, No. 5, pp. 79–85, 1991.

[37] S. Gottschalk, M. C. Lin, and D. Manocha. Obbtree: a hierarchical structure for rapid interference detection. In *SIGGRAPH '96: Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pp. 171–180, New York, NY, USA, 1996. ACM.

[38] R. L. Graham. An efficient algorith for determining the convex hull of a finite planar set. *Information Processing Letters*, Vol. 1, No. 4, pp. 132 – 133, 1972.

[39] Eran Guendelman, Robert Bridson, and Ronald Fedkiw. Nonconvex rigid bodies with stacking. In *SIGGRAPH '03: ACM SIGGRAPH 2003 Papers*, pp. 871–878, New York, NY, USA, 2003. ACM.

[40] Toshiya Hachisuka, Shinji Ogaki, and Henrik Wann Jensen. Progressive photon mapping. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, pp. 1–8, New York, NY, USA, 2008. ACM.

[41] Takahiro Harada, Seiichi Koshizuka, and Yoichiro Kawaguchi. Sliced data structure for particle-based simulations on gpus. In *GRAPHITE '07: Proceedings of the 5th international conference on Computer graphics and interactive techniques in Australia and Southeast Asia*, pp. 55–62, New York, NY, USA, 2007. ACM.

[42] Takahiro Harada, Seiichi Koshizuka, and Yoichiro Kawaguchi. Smoothed particle hydrodynamics on gpus. pp. 63–70, 2007.

[43] M. Harris, S. Sengupta, and J. D. Owens. Parallel prefix sum (scan) with cuda. In H. Nguyen, editor, *GPU Gems 3*. Addison Wesley, Aug 2007.

[44] Paul S. Heckbert. Discontinuity meshing for radiosity. In *Third Eurographics Workshop on Rendering*, pp. 203–226, 1992.

[45] L. Hernquist and N. Katz. TREESPH - A unification of SPH with the hierarchical tree method. *The Astrophysical Journal Supplement Series*, Vol. 70, pp. 419–446, June 1989.

[46] Robert Herzog, Vlastimil Havran, Shinichi Kinuwaki, Karol Myszkowski, and Hans-Peter Seidel. Global illumination using photon ray splatting. *Computer Graphics Forum*, Vol. 26, pp. 503–513(11), September 2007.

[47] Philip M. Hubbard. Real-time collision detection and time-critical computing. In *In Workshop On Simulation and Interation in Virtual Environments*, pp. 92–96, 1995.

[48] Frederik W Jansen. Data structures for ray tracing. In *Proceedings of a workshop (Eurographics Seminars on Data structures for raster graphics*, pp. 57–73, New York, NY, USA, 1986. Springer-Verlag New York, Inc.

[49] Maarten Jansen, Guy P. Nason, and Bernard W. Silverman. Scattered data smoothing by empirical bayesian shrinkage of second-generation wavelet coefficients. In Andrew F. Laine, Michael A. Unser, and Akram Aldroubi, editors, *Wavelets: Applications in Signal and Image Processing IX*, Vol. 4478, pp. 87–97. SPIE, 2001.

[50] Henrik Wann Jensen. Global illumination using photon maps. In *Proceedings of the eurographics workshop on Rendering techniques '96*, pp. 21–30, London, UK, 1996. Springer-Verlag.

[51] David A. J. Jevans. Adaptive voxel subdivision for ray tracing. In *Graphics Interface '89*, pp. 164–172, 1989.

[52] James T. Kajiya. The rendering equation. *SIGGRAPH Comput. Graph.*, Vol. 20, No. 4, pp. 143–150, 1986.

[53] I. Kamel and C. Faloutsos. Hilbert r-tree: An improved r-tree using fractals. In *Proceeding of VLDB Conference*, pp. 500–509, 1994.

[54] M.R. Kaplan. The use of spatial coherence in ray tracing. In *ACM SIGGRAPH Course Notes 11*, 1985.

[55] Douglas Scott Kay and Donald Greenberg. Transparency for computer synthesized images. *SIGGRAPH Comput. Graph.*, Vol. 13, No. 2, pp. 158–164, 1979.

[56] Timothy L. Kay and James T. Kajiya. Ray tracing complex scenes. *SIGGRAPH Comput. Graph.*, Vol. 20, No. 4, pp. 269–278, 1986.

[57] David G. Kirkpatrick and Raimund Seidel. The ultimate planar convex hull algorithm ? Technical report, Ithaca, NY, USA, 1983.

[58] James T. Klosowski, Martin Held, Joseph S.B. Mitchell, Henry Sowizral, and Karel Zikan. Efficient collision detection using bounding volume hierarchies of k-dops. *IEEE Transactions on Visualization and Computer Graphics*, Vol. 4, pp. 21–36, 1998.

[59] Eric Lafortune and Yves D. Willems. Bi-directional path tracing. Proceedings of Third International Conference on Computational Graphics and Visualization Techniques (Compugraphics ' 93, pp. 145–153, 1993.

[60] Y.-C. Lai, S. Fan, S. Chenney, and C. R. Dyer. Photorealistic image rendering with popu-

lation monte carlo energy redistribution. In *Proc. Eurographics Symposium on Rendering*, pp. 287–296, 2007.

[61] Elmar Langetepe, Gabriel Zachmann, 浩中本. 空間的データ構造とアルゴリズム. ボーンデジタル, 2007.

[62] Thomas Larsson and Tomas Akenine-Möller. Collision detection for continuously deforming bodies. In *Eurographics 2001, Short Presentations*, pp. 325–333. Eurographics Association, September 2001.

[63] C. Lauterbach, M. Garland, S. Sengupta, D. Luebke, and D. Manocha. Fast bvh construction on gpus. *Computer Graphics Forum*, Vol. 28, No. 2, pp. 375–384, 2009.

[64] S. Lefebvre, S. abd Hornus and F Neyret. Octree textures on the gpu. In *GPU Gems II*, chapter 37, pp. 595–613. NVIDIA, 2005.

[65] Scott T. Leutenegger, Jeffrey M. Edgington, and Mario A. Lopez. Str: A simple and efficient algorithm for r-tree packing. Technical report, 1997.

[66] C Levinthal. Molecular model-building by computer. *Scientific American*, Vol. 214, No. 6, pp. 42–52, Jun 1966.

[67] Ming C. Lin and John F. Canny. A fast algorithm for incremental distance calculation. In *In IEEE International Conference on Robotics and Automation*, pp. 1008–1014, 1991.

[68] Ming Chieh Lin. *Efficient collision detection for animation and robotics*. PhD thesis, 1993. Chair-Canny, John F.

[69] Peter Lindstrom, David Koller, William Ribarsky, Larry F. Hodges, Nick Faust, and Gregory A. Turner. Real-time, continuous level of detail rendering of height fields. pp. 109–118, 1996.

[70] Peter Lindstrom and Valerio Pascucci. Visualization of large terrains made easy. In *VIS '01: Proceedings of the conference on Visualization '01*, pp. 363–371, Washington, DC, USA, 2001. IEEE Computer Society.

[71] Stuart P. Lloyd. Least squares quantization in pcm. *IEEE Transactions on Information Theory*, Vol. 28, pp. 129–137, 1982.

[72] S.H. Lo. Volume discretization into tetrahedra-i. verification and orientation of boundary surfaces. *Computers and Structures*, Vol. 39, No. 5, pp. 493–500, 1991.

[73] R Löhner. Progress in grid generation via the advancing front technique. *Engineering with Computers (New York)*, Vol. 12, No. 3-4, pp. 186–210, 1996.

[74] J. David MacDonald and Kellogg S. Booth. Heuristics for ray tracing using space subdivision. *The Visual Computer*, Vol. 6, No. 3, pp. 153–166, 5 1990.

[75] Songrit Maneewongvatana and David M. Mount. It's okay to be skinny, if your friends are fat. In *Center for Geometric Computing 4th Annual Workshop on Computational Geometry*, 1999.

[76] J J Monaghan. Smoothed particle hydrodynamics. *Annual Review of Astronomy and Astrophysics*, Vol. 30, No. 1, pp. 543–574, 1992.

[77] D. E. Muller and F. P. Preparata. Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, Vol. 7, No. 2, pp. 217 – 236, 1978.

[78] Gordon Müller, Stephan Schäfer, and Dieter W. Fellner. Automatic creation of object hierarchies for radiosity clustering. In *PG '99: Proceedings of the 7th Pacific Conference on Computer Graphics and Applications*, p. 21, Washington, DC, USA, 1999. IEEE Computer Society.

[79] Matthias Müller, David Charypar, and Markus Gross. Particle-based fluid simulation for interactive applications. pp. 154–159, 2003.

[80] F. K. Musgrave, C. E. Kolb, and R. S. Mace. The synthesis and rendering of eroded fractal terrains. In *SIGGRAPH '89: Proceedings of the 16th annual conference on Computer graphics and interactive techniques*, pp. 41–50, New York, NY, USA, 1989. ACM.

[81] Bruce Naylor, John Amanatides, and William Thibault. Merging bsp trees yields polyhedral set operations. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques*, pp. 115–124, New York, NY, USA, 1990. ACM.

[82] Tomoyuki Nishita and Eihachiro Nakamae. Continuous tone representation of three-dimensional objects taking account of shadows and interreflection. *SIGGRAPH Comput. Graph.*, Vol. 19, No. 3, pp. 23–30, 1985.

[83] J. F. O'Brien and J. K. Hodgins. Dynamic simulation of splashing fluids. In *CA '95: Proceedings of the Computer Animation*, p. 198, Washington, DC, USA, 1995. IEEE Computer Society.

[84] I. J. Palmer and R. L. Grimsdale. Collision detection for animation using sphere-trees. *Computer Graphics Forum*, Vol. 14, No. 2, pp. 105–116, 1995.

[85] Michael S. Paterson and F. Frances Yao. Efficient binary space partitions for hidden-surface removal and solid modeling. *Discrete and Computational Geometry*, Vol. 5, No. 1, pp.

485–503, 12 1990.

[86] Madhav Ponamgi, Dinesh Manocha, and Ming C. Lin. Incremental algorithms for collision detection between solid models. In *SMA '95: Proceedings of the third ACM symposium on Solid modeling and applications*, pp. 293–304, New York, NY, USA, 1995. ACM.

[87] Stefan Popov, Johannes Günther, Hans-Peter Seidel, and Philipp Slusallek. Stackless kd-tree traversal for high performance GPU ray tracing. *Computer Graphics Forum*, Vol. 26, No. 3, pp. 415–424, September 2007. (Proceedings of Eurographics).

[88] F. P. Preparata and S. J. Hong. Convex hulls of finite sets of points in two and three dimensions. *Commun. ACM*, Vol. 20, No. 2, pp. 87–93, 1977.

[89] Timothy J. Purcell, Ian Buck, William R. Mark, and Pat Hanrahan. Ray tracing on programmable graphics hardware. In *SIGGRAPH '05: ACM SIGGRAPH 2005 Courses*, p. 268, New York, NY, USA, 2005. ACM.

[90] Timothy J. Purcell, Craig Donner, Mike Cammarano, Henrik Wann Jensen, and Pat Hanrahan. Photon mapping on programmable graphics hardware. In *HWWS '03: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*, pp. 41–50, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[91] P. W. Randles and L. D. Libersky. Smoothed particle hydrodynamics: Some recent improvements and applications. *Computer Methods in Applied Mechanics and Engineering*, Vol. 139, No. 1-4, pp. 375 – 408, 1996.

[92] Zhong Ren, Rui Wang, John Snyder, Kun Zhou, Xinguo Liu, Bo Sun, Peter-Pike Sloan, Hujun Bao, Qunsheng Peng, and Baining Guo. Real-time soft shadows in dynamic scenes using spherical harmonic exponentiation. *ACM Trans. Graph.*, Vol. 25, No. 3, pp. 977–986, 2006.

[93] J. Revelles, C. Urena, and M. Lastra. An efficient parametric algorithm for octree traversal. In *Journal of WSCG*, pp. 212–219, 2000.

[94] Nick Roussopoulos and Daniel Leifker. Direct spatial search on pictorial databases using packed r-trees. In *SIGMOD '85: Proceedings of the 1985 ACM SIGMOD international conference on Management of data*, pp. 17–31, New York, NY, USA, 1985. ACM.

[95] Steven M. Rubin and Turner Whitted. A 3-dimensional representation for fast rendering of complex scenes. In *SIGGRAPH '80: Proceedings of the 7th annual conference on Computer graphics and interactive techniques*, pp. 110–116, New York, NY, USA, 1980.

ACM.

[96] Szymon Rusinkiewicz and Marc Levoy. Qsplat: a multiresolution point rendering system for large meshes. In *SIGGRAPH '00: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pp. 343–352, New York, NY, USA, 2000. ACM Press/Addison-Wesley Publishing Co.

[97] Nadathur Satish, Mark Harris, and Michael Garland. Designing efficient sorting algorithms for manycore gpus. NVIDIA Technical Report NVR-2008-001, NVIDIA Corporation, September 2008.

[98] T. Sellis, N. Roussopoulos, and C. Faloutsos. The r+-tree: A dynamic index for multi-dimensional objects, 1987.

[99] Maxim Shevtsov, Alexei Soupikov, and Alexander Kapustin. Highly parallel fast kd-tree construction for interactive ray tracing of dynamic scenes. *Comput. Graph. Forum*, Vol. 26, No. 3, pp. 395–404, 2007.

[100] Jonathan Richard Shewchuk. Delaunay refinement algorithms for triangular mesh generation. *Computational Geometry*, Vol. 22, No. 1-3, pp. 21 – 74, 2002.

[101] Peter-Pike Sloan, Jan Kautz, and John Snyder. Precomputed radiance transfer for real-time rendering in dynamic, low-frequency lighting environments. In *SIGGRAPH '02: Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pp. 527–536, New York, NY, USA, 2002. ACM.

[102] Martin Stich, Heiko Friedrich, and Andreas Dietrich. Spatial splits in bounding volume hierarchies. In *HPG '09: Proceedings of the Conference on High Performance Graphics 2009*, pp. 7–13, New York, NY, USA, 2009. ACM.

[103] Matthias Teschner, Bruno Heidelberger, Matthias Mueller, Danat Pomeranets, and Markus Gross. Optimized spatial hashing for collision detection of deformable objects. In *Proceedings of VMV'03*, pp. 47–54, 2003.

[104] John A. Tsakok. Faster incoherent rays: Multi-bvh ray stream tracing. In *HPG '09: Proceedings of the Conference on High Performance Graphics 2009*, pp. 151–158, New York, NY, USA, 2009. ACM.

[105] Gino van den Bergen. Efficient collision detection of complex deformable models using aabb trees. *J. Graph. Tools*, Vol. 2, No. 4, pp. 1–13, 1997.

[106] Eric Veach and Leonidas Guibas. Bidirectional estimators for light transport. In *Proc. 5th*

Eurographics Workshop on Rendering, pp. 147–162, 1994.

[107] Eric Veach and Leonidas J. Guibas. Metropolis light transport. In *SIGGRAPH '97: Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pp. 65–76, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[108] J. Villard and H. Borouchaki. Adaptive meshing for cloth animation. *Eng. with Comput.*, Vol. 20, No. 4, pp. 333–341, 2005.

[109] Carsten Wachter and Er Keller. Instant ray tracing: The bounding interval hierarchy. In *In Rendering Techniques 2006 Proceedings of the 17th Eurographics Symposium on Rendering*, pp. 139–149, 2006.

[110] I. Wald. On fast construction of sah-based bounding volume hierarchies. pp. 33–40, Sept. 2007.

[111] D. F. Watson. Computing the n-dimensional Delaunay tessellation with application to Voronoi polytopes. *The Computer Journal*, Vol. 24, No. 2, pp. 167–172, 1981.

[112] Turner Whitted. An improved illumination model for shaded display. *Commun. ACM*, Vol. 23, No. 6, pp. 343–349, 1980.

[113] Sven Woop, Jörg Schmittler, and Philipp Slusallek. Rpu: a programmable ray processing unit for realtime ray tracing. *ACM Trans. Graph.*, Vol. 24, No. 3, pp. 434–444, 2005.

[114] Brian Wyvill. 3d grid hashing function. pp. 343–345, 1990.

[115] Ren Yasuda, Takahiro Harada, and Yoichiro Kawaguchi. Fast rendering of particle-based fluid by utilizing simulation data. *Eurographics Short paper*, 2009.

[116] Mark A. Yerry and Mark S. Shephard. Automatic three-dimensional mesh generation by the modified-octree technique. *International Journal for Numerical Methods in Engineering*, Vol. 20, No. 11, pp. 1965–1990, 1984.

[117] G. Zachmann. Rapid collision detection by dynamically aligned dop-trees. *Virtual Reality Annual International Symposium*, Vol. 0, p. 90, 1998.

[118] Gabriel Zachmann. Real-time and exact collision detection for interactive virtual prototyping. In *The 1997 ASME Design Engineering Technical Conferences*, September 1997.

[119] Gabriel Zachmann. Minimal hierarchical collision detection. In *VRST '02: Proceedings of the ACM symposium on Virtual reality software and technology*, pp. 121–128, New York, NY, USA, 2002. ACM.

[120] Kun Zhou, Qiming Hou, Rui Wang, and Baining Guo. Real-time kd-tree construction on

graphics hardware. In *SIGGRAPH Asia '08: ACM SIGGRAPH Asia 2008 papers*, pp. 1–11, New York, NY, USA, 2008. ACM.

付録 A

略語一覧

3D DDA	3-dimensional Digital Differential Analyzer; 3 次元デジタル微分解析器	22
3DCG	3 Dimensional Computer Graphics; 3 次元コンピュータグラフィクス	7
AABB	Axis-Aligned Bounding Box	29
AR	Augumented Reality; 拡張現実	2
BB	Branch and Bound; 分枝限定法	6
BDPT	Bidirectional Path Tracing; 双方向経路追跡	13
BIH	Bounding Internal Hierarchy	31
BSP	Binary Space Partitioning; 二分空間分割	8
BVH	Bounding Volume Hierarchy	9
CAD	Computer Aided Design; コンピュータ支援設計	2
CAE	Computer Aided Engineering; コンピュータ支援工学	3
CAM	Computer Aided Manufacturing; コンピュータ支援製造	2
Cell BE	Cell Broadband Engine	19
CG	Computer Graphics; コンピュータ グラフィクス	2
CNRS	Centre National de la Recherche Scientifique; フランス国立科学研究センター	65
CPU	Central Processing Unit; 中央処理装置	2
CSG	Constructive Solid Geometry	8
CTO	Chief Technology Officer, Chief Technical Officer; 最高技術責任者	64
CUDA	Compute Unified Device Architecture	64
DBVH	Distorted Bounding Volume Hierarchy	3
DF	Distance Filed; 距離場	10

ERPT Energy Redistribution Path Tracing; エネルギー再分布経路追跡 13

fBm fractional Brownian motion; フラクタル ブラウン運動 8

FDM Finite Difference Method; 有限差分法 11

FEM Finite Element Method; 有限要素法 1

FPGA Field Programmable Gate Array 28

fps Frame Per Second 28

FVM Finite Volume Method; 有限体積法 11

GA Genetic Algorithm; 遺伝的アルゴリズム 6

GI Global Illumination; 大域照明, グローバル イルミネーション 13

GJK Gilbert-Johnson-Keerthi 10

GPU Graphics Processing Unit; グラフィクス プロセシング ユニット 2

HPC High-Performance Computing 2

IRDVC Inverse Ray Direction Vector Components 46

k-DOP k-Discrete Oriented Polytope 29

kd-tree K-Dimentional Tree; kd 木 13

KNNS K-Nearest Neighbor Search; k 近傍探索 6

LAAS Laboratoire d'Analyse et d'Architecture des Systemes; システムアーキテクチャ/解析研究所 65

LBVH Linear Bounding Volume Hierarcy 31

LOD Level Of Detail 17

LSS Line Swept Sphere, capsule 29

MBR Minimum Bounding Rectangle 29

MCPT Monte-Carlo Path Tracing; モンテカルロ経路追跡 13

MLT Metropolis Light Transport; メトロポリス光輸送 13

MLT Metropolis light transport; メトロポリス光輸送 16

NNS Nearest Neighbor Search; 最近傍探索 6

NP Non-deterministic Polynomial-time; 非決定論的な多項式時間 6

NURBS Non-Uniform Rational B-Spline; 非一様有理 B スプライン 7

OBB Oriented Bounding Box 29

PMC-ER Population Monte-Carlo Energy Redistribution; ポピュレーション モンテカルロ エネルギー再分布 13

PPU	PowerPC Processing Unit	19
PRT	Precomputed Radiance Transfer; 事前計算放射輝度伝播	16
QBVH	Quadric Bounding Volume Hierarchy	31
RPU	Ray Processing Unit	28
SAH	Surface Area Heuristic	26
SBVH	Split Bounding Volume Hierarchy	31
SCE	Sony Computer Entertainment	19
SIMD	Single Instruction Multiple Data	31
SPH	Smoothed Particle Dynamics	11
SPT	Sequantial Point Tree	17
SPU	Synergistic Processing Unit	19
SSE	Streaming SIMD Extension	31
STL	Standard Template Library	39
SVM	Support Vector Machine; サポート ベクター マシン	1
TIN	Triangular Irregular Network	8
TNP	Traveling Salesman Problem; 巡回セールスマン問題	6
VLSI	Very Large Scale Integration	3
VVH	Voxel Volume Heuristic	26

以上

1 p ～ 96 p 完

修士学位論文

平成 21 年度

東京大学大学院 学際情報学府先端表現情報学 コース
086302 今村 紀之